

UnifiedPOS

UnifiedPOS Retail Peripheral Architecture

Version 1.8

June 30, 2003

International Standard

For Implementation of Point Of Service Peripherals

Copyright © National Retail Federation, 2003. All Rights Reserved.

Right to Copy

This document may be copied or used for purposes consistent with adoption of the ARTS Standards. However, any changes or inconsistent uses must be pre-approved in writing by the National Retail Federation (“NRF”). Consequently, this document may be furnished to others, but derivative works (the term “derivative works” does not include functional additions that do not modify or change the base standard as written) that comment on or otherwise explain it or assist in its implementation may not cite or refer to the standard, in whole or in part, without such permission. Moreover, this document may not be modified in any way, such as by removing the copyright notice or references to the NRF, ARTS, or its committees, except as needed for the purpose of developing ARTS standards using procedures approved by NRF, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the National Retail Federation or its successors or assigns.

Disclaimer

This document and the information contained herein is provided on an “AS IS” basis and THE NATIONAL RETAIL FEDERATION (NRF) DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

THE NATIONAL RETAIL FEDERATION (NRF) ASSUMES NO RESPONSIBILITY FOR ERRORS OR OMISSIONS IN THIS PUBLICATION OR OTHER DOCUMENTS WHICH ARE REFERENCED BY, CITED BY, OR LINKED TO THIS PUBLICATION. THIS PUBLICATION COULD INCLUDE TECHNICAL OR OTHER INACCURACIES OR TYPOGRAPHICAL ERRORS. THE NATIONAL RETAIL FEDERATION (NRF) RESERVES THE RIGHT TO MAKE IMPROVEMENTS AND/OR CHANGES TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW VERSIONS PUBLISHED FROM TIME TO TIME AS DETERMINED BY THE NATIONAL RETAIL FEDERATION (NRF).

UnifiedPOS Technical Committee Members:

**360Commerce,
Axiohm Inc.,
Fujitsu Transaction Solutions Inc.,
The Home Depot, Inc.,
IBM Corporation,
Microsoft Corporation,
NCR Corporation,
PCMS Datafit Ltd.,
J.C. Penney Company, Inc.,
Research Computer Services, Inc.,
Sears, Roebuck & Co.,
Seiko Epson Corporation,
Sun Microsystems, Inc.,
Ultimate Technology Corporation,
Wincor Nixdorf International 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.

This page intentionally left blank.

Table of Contents

INTRODUCTION AND ARCHITECTURE	
UNIFIEDPOS ARCHITECTURE FOR RETAIL	1
WHAT IS UNIFIEDPOS?	1
GOALS	3
DEPENDENCIES	3
UNIFIEDPOS RELATIONSHIP TO OPOS AND JAVAPOS	3
WHO SHOULD READ THIS DOCUMENT	4
ARCHITECTURAL OVERVIEW	5
ARCHITECTURAL COMPONENTS	5
USE OF UML	6
<i>Package Diagram</i>	8
DATA TYPES	9
DEVICE BEHAVIOR MODELS	10
INTRODUCTION TO PROPERTIES, METHODS, AND EVENTS	10
<i>Properties (UML Attributes)</i>	10
<i>Methods (UML Operations)</i>	11
<i>Events (UML Interfaces)</i>	11
DEVICE INITIALIZATION AND FINALIZATION.....	12
<i>Initialization</i>	12
<i>Finalization</i>	12
<i>Summary</i>	13
DEVICE SHARING MODEL.....	14
<i>Exclusive-Use Devices</i>	14
<i>Sharable Devices</i>	14
EVENTS	15
ERRORS.....	16
ERROR CODES.....	16
<i>Extended Error Code</i>	17
DEVICE INPUT MODEL.....	18
<i>Error Handling</i>	19
<i>Miscellaneous</i>	20
DEVICE OUTPUT MODELS	21
<i>Synchronous Output</i>	21
<i>Asynchronous Output</i>	21
DEVICE POWER REPORTING MODEL	22
<i>Model</i>	22
<i>Power State Diagram</i>	23
<i>Power Properties</i>	24
<i>Power Reporting Requirements for DeviceEnabled</i>	25
DEVICE INFORMATION REPORTING MODEL	26
<i>Statistics Reporting Properties and Methods</i>	26
<i>XML definitions for POS Device Statistics</i>	27
DEVICE STATES	29
<i>Device State Diagram</i>	30
VERSION HANDLING.....	31

CHAPTER 1	
COMMON PROPERTIES, METHODS, AND EVENTS	33
SUMMARY	33
GENERAL INFORMATION.....	36
PROPERTIES (UML ATTRIBUTES)	38
METHODS (UML OPERATIONS)	49
EVENTS (UML INTERFACES).....	57
CHAPTER 2	
BUMP BAR	65
SUMMARY	65
GENERAL INFORMATION.....	69
<i>Capabilities</i>	69
<i>Bump Bar Class Diagram</i>	70
<i>Model</i>	71
Input – Bump Bar	72
Output – Tone.....	73
<i>Device Sharing</i>	73
<i>Bump Bar State Diagram</i>	74
PROPERTIES (UML ATTRIBUTES)	75
METHODS (UML OPERATIONS)	81
EVENTS (UML INTERFACES).....	86
CHAPTER 3	
CASH CHANGER.....	91
SUMMARY	91
GENERAL INFORMATION.....	95
<i>Capabilities</i>	95
<i>CashChanger Class Diagram</i>	96
<i>Model</i>	97
<i>Cash Changer Sequence Diagram</i>	100
<i>Cash Changer State Diagram</i>	101
<i>Device Sharing</i>	101
PROPERTIES (UML ATTRIBUTES)	102
METHODS (UML OPERATIONS)	112
EVENTS (UML INTERFACES).....	119
CHAPTER 4	
CASH DRAWER	121
SUMMARY	121
GENERAL INFORMATION.....	124
<i>Capabilities</i>	124
<i>Cash Drawer Class Diagram</i>	124
<i>Cash Drawer Sequence Diagram</i>	125
<i>Device Sharing</i>	126
PROPERTIES (UML ATTRIBUTES)	127
METHODS (UML OPERATIONS)	129
EVENTS (UML INTERFACES).....	130

CHAPTER 5	
CAT - CREDIT AUTHORIZATION TERMINAL	133
SUMMARY	133
GENERAL INFORMATION.....	137
<i>Description of terms</i>	137
<i>Capabilities</i>	138
<i>CAT Class Diagram</i>	139
<i>Model</i>	140
<i>Device Sharing</i>	143
<i>CAT Sequence Diagram</i>	144
<i>CAT State Diagram</i>	145
PROPERTIES (UML ATTRIBUTES)	146
METHODS (UML OPERATIONS)	163
EVENTS (UML INTERFACES)	171
CHAPTER 6	
CHECK SCANNER	175
SUMMARY	175
GENERAL INFORMATION.....	179
<i>Capabilities</i>	179
<i>Check Scanner Class Diagram</i>	180
<i>Model</i>	181
<i>Device Sharing</i>	184
<i>Check Scanner Sequence Diagram</i>	185
<i>Check Scanner State Diagram</i>	186
PROPERTIES (UML ATTRIBUTES)	187
METHODS (UML OPERATIONS)	201
EVENTS (UML INTERFACES)	210
CHAPTER 7	
COIN DISPENSER.....	213
SUMMARY	213
GENERAL INFORMATION.....	216
<i>Capabilities</i>	216
<i>Coin Dispenser Class Diagram</i>	217
<i>Coin Dispenser Sequence Diagram</i>	218
<i>Model</i>	219
<i>Device Sharing</i>	219
PROPERTIES (UML ATTRIBUTES)	220
METHODS (UML OPERATIONS)	221
EVENTS (UML INTERFACES)	222
CHAPTER 8	
FISCAL PRINTER.....	225
SUMMARY	225
GENERAL INFORMATION.....	234
<i>Fiscal Printer Class Diagram</i>	235
<i>General Requirements</i>	236
<i>Fiscal Printer Modes</i>	237
<i>Model</i>	238
<i>Error Model</i>	239

<i>Release 1.8 additional Model clarifications</i>	240
<i>Fiscal Printer States</i>	242
<i>Document Printing</i>	244
<i>Ordering of Fiscal Receipt Print Requests</i>	245
<i>Fiscal Receipt Layouts</i>	247
<i>Example of a Fiscal Receipt</i>	248
<i>Totalizers and Fiscal Memory</i>	249
<i>Counters</i>	249
<i>VAT Tables</i>	249
<i>Receipt Duplication</i>	249
<i>Currency amounts, percentage amounts, VAT rates, & quantity amounts</i> ..	250
<i>Currency Change</i>	250
<i>Device Sharing</i>	250
PROPERTIES (UML ATTRIBUTES)	251
METHODS (UML OPERATIONS)	289
EVENTS (UML INTERFACES)	363
CHAPTER 9	
HARD TOTALS	369
SUMMARY	369
GENERAL INFORMATION	373
<i>Capabilities</i>	373
<i>Hard Totals Class Diagram</i>	374
<i>Hard Totals Sequence Diagram</i>	375
<i>Model</i>	376
<i>Device Sharing</i>	378
PROPERTIES (UML ATTRIBUTES)	379
METHODS (UML OPERATIONS)	381
EVENTS (UML INTERFACES)	391
CHAPTER 10	
KEYLOCK	393
SUMMARY	393
GENERAL INFORMATION	396
<i>Capabilities</i>	396
<i>Keylock Class Diagram</i>	396
<i>Keylock Sequence Diagram</i>	397
<i>Model</i>	398
<i>Device Sharing</i>	398
PROPERTIES (UML ATTRIBUTES)	399
METHODS (UML OPERATIONS)	400
EVENTS (UML INTERFACES)	401
CHAPTER 11	
LINE DISPLAY	403
SUMMARY	403
GENERAL INFORMATION	407
<i>Capabilities</i>	407
<i>Line Display Class Diagram</i>	408
<i>Line Display Sequence Diagram</i>	409
<i>Model</i>	410
<i>Display Modes</i>	411

<i>Data Characters and Escape Sequences</i>	412
<i>Device Sharing</i>	412
PROPERTIES (UML ATTRIBUTES)	413
METHODS (UML OPERATIONS)	434
EVENTS (UML INTERFACES)	449
CHAPTER 12	
MICR - MAGNETIC INK CHARACTER RECOGNITION READER	451
SUMMARY	451
GENERAL INFORMATION.....	454
<i>Capabilities</i>	454
<i>MICR Class Diagram</i>	455
<i>MICR Sequence Diagram</i>	456
<i>Model</i>	457
<i>Device Sharing</i>	458
<i>MICR Character Substitution</i>	459
PROPERTIES (UML ATTRIBUTES)	460
METHODS (UML OPERATIONS)	464
EVENTS (UML INTERFACES)	468
CHAPTER 13	
MOTION SENSOR	473
SUMMARY	473
GENERAL INFORMATION.....	475
<i>Capabilities</i>	475
<i>Motion Sensor Class Diagram</i>	475
<i>Model</i>	476
<i>Device Sharing</i>	476
<i>Motion Sensor Sequence Diagram</i>	477
<i>Motion Sensor State Diagram</i>	478
PROPERTIES (UML ATTRIBUTES)	479
METHODS (UML OPERATIONS)	480
EVENTS (UML INTERFACES)	481
CHAPTER 14	
MSR - MAGNETIC STRIPE READER	483
SUMMARY	483
GENERAL INFORMATION.....	486
<i>Capabilities</i>	486
Clarifications for JIS-II data handling.....	486
<i>MSR Class Diagram</i>	487
<i>Device Behavior Model</i>	488
Input – MSR	488
<i>Device Sharing</i>	488
<i>MSR Sequence Diagram</i>	489
<i>MSR State Diagrams</i>	490
PROPERTIES (UML ATTRIBUTES)	492
EVENTS (UML INTERFACES)	503

CHAPTER 15	
PIN PAD	509
SUMMARY	509
GENERAL INFORMATION.....	513
<i>Capabilities</i>	513
<i>PIN Pad Class Diagram</i>	514
<i>PIN Pad Sequence Diagram</i>	515
<i>Feature Not Supported</i>	516
<i>Note on Terminology</i>	516
<i>Model</i>	517
<i>Device Sharing</i>	517
<i>PIN Pad State Diagram</i>	518
PROPERTIES (UML ATTRIBUTES)	519
METHODS (UML OPERATIONS)	530
EVENTS (UML INTERFACES).....	535
CHAPTER 16	
POINT CARD READER WRITER.....	539
SUMMARY	539
GENERAL INFORMATION.....	544
<i>Capabilities</i>	544
<i>Point Card Reader Writer Class Diagram</i>	545
<i>Model</i>	546
Input Model	546
Output Model.....	547
<i>Card Insertion Diagram</i>	548
<i>Printing Capability</i>	549
<i>Cleaning Capability</i>	550
<i>Initialization of Magnetic Stripe Data</i>	550
<i>Device Sharing</i>	550
<i>Data Characters and Escape Sequences</i>	551
<i>Point Card Reader Writer Sequence Diagram</i>	553
<i>Point Card Reader Writer State Diagram</i>	554
PROPERTIES (UML ATTRIBUTES).....	555
METHODS (UML OPERATIONS)	576
EVENTS (UML INTERFACES).....	584
CHAPTER 17	
POS KEYBOARD.....	589
SUMMARY	589
GENERAL INFORMATION.....	592
<i>Capabilities</i>	592
<i>POS Keyboard Class Diagram</i>	592
<i>POS Keyboard Sequence Diagram</i>	593
<i>Model</i>	594
Keyboard Translation	594
<i>Device Sharing</i>	594
PROPERTIES (UML ATTRIBUTES)	595
EVENTS (UML INTERFACES).....	597

CHAPTER 18	
POS POWER.....	601
SUMMARY	601
GENERAL INFORMATION.....	604
<i>Capabilities</i>	604
<i>Device Sharing</i>	604
<i>Model</i>	605
<i>POSPower Class Diagram</i>	606
<i>POSPower Sequence Diagram</i>	607
<i>POSPower State Diagram</i>	608
<i>POSPower PowerState Diagram - part 1</i>	609
<i>POSPower PowerState Diagram - part 2</i>	610
<i>POSPower PowerState Diagram - part 3</i>	611
<i>POSPower State chart Diagram for Fan and Temperature</i>	612
PROPERTIES (UML ATTRIBUTES)	613
METHODS (UML OPERATIONS)	617
EVENTS (UML INTERFACES).....	618
CHAPTER 19	
POS PRINTER.....	621
SUMMARY	621
GENERAL INFORMATION.....	627
<i>Capabilities</i>	627
<i>POS Printer Class Diagram</i>	628
<i>POS Printer Class Diagram Updates</i>	629
<i>Model</i>	630
<i>Device Sharing</i>	635
<i>POS Printer State Diagram</i>	636
<i>“Both sides printing” sequence Diagram</i>	637
<i>Data Characters and Escape Sequences</i>	638
<i>POS Printer State Diagrams (Low Level)</i>	641
PROPERTIES (UML ATTRIBUTES)	646
METHODS (UML OPERATIONS)	691
EVENTS (UML INTERFACES).....	723
CHAPTER 20	
REMOTE ORDER DISPLAY	729
SUMMARY	729
GENERAL INFORMATION.....	733
<i>Capabilities</i>	733
<i>Remote Order Display Class Diagram</i>	734
<i>Model</i>	735
<i>Device Sharing</i>	739
PROPERTIES (UML ATTRIBUTES)	740
METHODS (UML OPERATIONS)	751
EVENTS (UML INTERFACES).....	768
CHAPTER 21	
SCALE	773
SUMMARY	773
GENERAL INFORMATION.....	776

<i>Capabilities</i>	776
<i>Scale Class Diagram</i>	777
<i>Scale Sequence Diagram</i>	778
<i>Model</i>	779
<i>Device Sharing</i>	779
PROPERTIES (UML ATTRIBUTES)	780
METHODS (UML OPERATIONS)	785
EVENTS (UML INTERFACES)	788
CHAPTER 22	
SCANNER (BAR CODE READER).....	791
SUMMARY	791
GENERAL INFORMATION.....	794
<i>Capabilities</i>	794
<i>Scanner Class Diagram</i>	794
<i>Scanner Sequence Diagram</i>	795
<i>Model</i>	796
<i>Device Sharing</i>	796
PROPERTIES (UML ATTRIBUTES)	797
EVENTS (UML INTERFACES).....	802
CHAPTER 23	
SIGNATURE CAPTURE	805
SUMMARY	805
GENERAL INFORMATION.....	808
<i>Capabilities</i>	808
<i>Signature Capture Class Diagram</i>	809
<i>Signature Capture Sequence Diagram</i>	810
<i>Model</i>	811
<i>Device Sharing</i>	812
PROPERTIES (UML ATTRIBUTES)	813
METHODS (UML OPERATIONS)	817
EVENTS (UML INTERFACES).....	819
CHAPTER 24	
SMART CARD READER / WRITER.....	823
SUMMARY	823
GENERAL INFORMATION.....	827
<i>Capabilities</i>	827
<i>Smart Card Reader / Writer Class Diagram</i>	828
<i>Model</i>	829
<i>Card Insertion Diagram</i>	832
<i>Device Sharing</i>	833
<i>Data Transfer Modes</i>	834
<i>Smart Card Reader / Writer Sequence Diagram</i>	835
<i>Smart Card Reader / Writer State Diagram</i>	836
PROPERTIES (UML ATTRIBUTES).....	837
METHODS (UML OPERATIONS)	843
EVENTS (UML INTERFACES).....	848

CHAPTER 25	
 TONE INDICATOR.....	853
SUMMARY	853
GENERAL INFORMATION.....	856
<i>Capabilities</i>	856
<i>Tone Indicator Class Diagram</i>	856
<i>Tone Indicator Sequence Diagram</i>	857
<i>Model</i>	858
<i>Device Sharing</i>	859
PROPERTIES (UML ATTRIBUTES)	860
METHODS (UML OPERATIONS)	863
EVENTS (UML INTERFACES).....	865
APPENDIX A	
OLE FOR RETAIL POS — OPOS IMPLEMENTATION REFERENCE.....	1
WHAT IS “OLE FOR RETAIL POS?”	1
WHO SHOULD READ THIS SECTION	2
GENERAL OLE FOR RETAIL POS CONTROL MODEL	2
OPOS DEFINITIONS	3
<i>Device Class</i>	3
<i>Control Object or CO</i>	3
<i>Service Object or SO</i>	3
<i>OPOS Control or Control</i>	3
HOW AN APPLICATION USES AN OPOS CONTROL	4
WHEN METHODS AND PROPERTIES MAY BE ACCESSED	5
<i>Methods</i>	5
<i>Properties</i>	5
STATUS, RESULT CODE, AND STATE MODEL	7
<i>Status Model</i>	8
<i>Result Code Model</i>	8
<i>State Model</i>	9
DEVICE SHARING MODEL.....	10
<i>Exclusive-Use Devices</i>	10
<i>Sharable Devices</i>	10
EVENTS	11
<i>OPOS Event Registration Sequence Diagram</i>	13
INPUT MODEL	14
OUTPUT MODEL	16
<i>Synchronous Output</i>	16
<i>Asynchronous Output</i>	16
DEVICE POWER REPORTING MODEL	17
<i>Model</i>	17
<i>Properties</i>	18
<i>Power Reporting Requirements for DeviceEnabled</i>	19
DEVICE INFORMATION REPORTING MODEL	20
<i>Statistics Reporting Properties and Methods</i>	20
OPOS COMPONENT DESCRIPTIONS	21
SECTION 1: OPOS DATA TYPES.....	22
SECTION 2: OPOS INTERFACE DESCRIPTIONS	23
OPOS COMMON PROPERTIES, METHODS, AND EVENTS	24
<i>Common Properties</i>	24

<i>Common Methods</i>	25
<i>OPOS Programmatic Names</i>	26
<i>Properties</i>	27
<i>Methods</i>	44
<i>Events</i>	53
PERIPHERAL INTERFACES	57
OPOS: CASH DRAWER	58
<i>Visual Basic Command Examples</i>	58
<i>Initializing Properties, Methods, and Events</i>	58
<i>Capabilities, Assignments and Descriptions Properties, Methods, & Events</i>	58
<i>Cash Drawer Operations Properties and Methods</i>	59
<i>Terminating Methods</i>	59
<i>Visual C++ Command Examples</i>	60
<i>Initializing Properties, Methods, and Events</i>	60
<i>Capabilities, Assignments and Descriptions Properties, Methods, & Events</i>	60
<i>Cash Drawer Operations Properties and Methods</i>	61
<i>Terminating Methods</i>	61
OPOS: MICR	62
<i>Visual Basic Command Examples</i>	62
<i>Initializing Properties, Methods, and Events</i>	62
<i>Capabilities, Assignments and Descriptions Properties, Methods, & Events</i>	62
<i>MICR Operations Properties, Methods, and Events</i>	63
<i>Terminating Methods</i>	64
<i>Visual C++ Command Examples</i>	65
<i>Initializing Properties, Methods, and Events</i>	65
<i>Capabilities, Assignments and Descriptions Properties, Methods, & Events</i>	65
<i>MICR Operations Properties, Methods, and Events</i>	66
<i>Terminating Methods</i>	67
SECTION 3: OPOS REGISTRY USAGE	68
<i>Service Object Root Registry Key</i>	68
<i>Device Class Keys</i>	68
<i>Device Name Keys and Values</i>	69
<i>Logical Device Name Values</i>	69
<i>Service Provider Root Registry Key</i>	70
<i>Example</i>	70
SECTION 4: OPOS APPLICATION HEADER FILES	72
SECTION 5: TECHNICAL DETAILS	73
<i>System Strings (BSTR)</i>	73
System String Characteristics	73
System String Usage	73
<i>System Strings and Binary Data</i>	74
<i>Mapping of CharSet</i>	75
SECTION 6: RELEASE 1.5 API CHANGE: CLAIMDEVICE AND RELEASEDEVICE... ..	76
SECTION 7: OPOS APG CHANGE HISTORY	77
<i>Release 1.01</i>	77
<i>Release 1.1</i>	78
<i>Release 1.2</i>	80
<i>Release 1.3</i>	82
<i>Release 1.4</i>	84
<i>Release 1.5</i>	85
<i>Release 1.6</i>	87

<i>Release 1.7</i>	88
SECTION 8: OPOS CONTROL PROGRAMMER'S GUIDE	89
<i>Who Should Read This Section</i>	89
<i>General OLE for Retail POS Control Model</i>	90
<i>OPOS Definitions</i>	91
Device Class	91
Control Object or CO	91
Service Object or SO	91
OPOS Control or Control	91
<i>Interface Overview</i>	93
<i>Methods</i>	94
Open Method	94
Close Method.....	94
Other Methods	94
<i>Properties</i>	95
String Properties	95
LONG and BOOL Properties	95
Other Property Types.....	95
<i>Events</i>	96
Architecture: Firing an Event	96
Architectural Issue: Freezing Events by the Container	96
Architectural Feature: Freezing Events by the Application	97
Summary of Event Firing	97
<i>Control Object Responsibilities</i>	98
Methods	98
Properties	101
Events	102
<i>Service Object Responsibilities and Implementation</i>	106
Methods	106
Properties	113
Events	115
<i>OPOS CPG Change History</i>	116
Release 1.01	116
Release 1.1	116
Release 1.2	117
Release 1.3	118
Release 1.4	119
Release 1.5	119
Release 1.6	120
Release 1.7	120
<i>Common Control Objects</i>	121
Features.....	121
Availability and Future	121
<i>OPOS Internal Header Files</i>	122

APPENDIX B**JAVA FOR RETAIL POS — JAVAPOS IMPLEMENTATION REFERENCE 1**

WHAT IS JAVA FOR RETAIL POS?	1
BENEFITS	1
DEPENDENCIES	2
RELATIONSHIP TO OPOS	2
WHO SHOULD READ THIS SECTION	2
APPENDIX OVERVIEW	3
ARCHITECTURAL OVERVIEW	3
ARCHITECTURAL COMPONENTS	4
<i>Additional Layers and APIs</i>	5
<i>JavaPOS Development Environment</i>	5
DEVICE BEHAVIOR MODELS	6
INTRODUCTION TO PROPERTIES, METHODS, AND EVENTS	6
DEVICE INITIALIZATION AND FINALIZATION	7
<i>Initialization</i>	7
<i>Finalization</i>	7
<i>Summary</i>	8
DEVICE SHARING MODEL	9
<i>Exclusive-Use Devices</i>	10
<i>Sharable Devices</i>	10
DATA TYPES	11
EXCEPTIONS	12
<i>ErrorCode</i>	13
<i>ErrorCodeExtended</i>	14
EVENTS	15
<i>Registering for Events</i>	17
<i>Event Delivery</i>	17
<i>JavaPOS Event Registration Sequence Diagram</i>	18
DEVICE INPUT MODEL	19
<i>Error Handling</i>	20
<i>Miscellaneous</i>	21
DEVICE OUTPUT MODELS	22
<i>Synchronous Output</i>	22
<i>Asynchronous Output</i>	22
<i>Error Handling</i>	23
<i>Miscellaneous</i>	23
DEVICE POWER REPORTING MODEL	24
<i>Model</i>	24
<i>Properties</i>	25
<i>Power Reporting Requirements for DeviceEnabled</i>	26
DEVICE INFORMATION REPORTING MODEL	27
<i>Statistics Reporting Properties and Methods</i>	27
DEVICE STATES	28
THREADS	29
VERSION HANDLING	29
CLASSES AND INTERFACES	30
SYNOPSIS	30
<i>Application</i>	30
<i>Device Control</i>	31
<i>Device Service</i>	31

<i>Helper Classes</i>	32
SAMPLE CLASS AND INTERFACE HIERARCHIES.....	33
<i>Application Sample</i>	33
<i>Device Control Sample</i>	33
Scanner	33
POSPrinter	34
<i>Device Service Sample</i>	34
“MyScannerService”	34
“MyPrinterService”	35
SAMPLE APPLICATION CODE.....	36
PACKAGE STRUCTURE	37
<i>jpos</i>	37
<i>jpos.events</i>	40
<i>jpos.services</i>	41
DEVICE CONTROLS	43
DEVICE CONTROL RESPONSIBILITIES	43
DEVICE SERVICE MANAGEMENT.....	44
<i>jpos.config/loader (JCL) and JavaPOS Entry Registry (JER)</i>	44
<i>jpos.config/loader (JCL) Characteristics</i>	44
PROPERTY AND METHOD FORWARDING	47
EVENT HANDLING	48
<i>Event Listeners and Event Delivery</i>	48
<i>Event Callbacks</i>	49
DEVICE CONTROL VERSION HANDLING.....	50
DEVICE SERVICES.....	52
DEVICE SERVICE RESPONSIBILITIES	52
PROPERTY AND METHOD PROCESSING.....	52
EVENT GENERATION.....	53
PHYSICAL DEVICE ACCESS.....	54
API MAPPING RULES	54
JAVAPOS COMPONENT DESCRIPTIONS	55
SECTION 1: JAVAPOS DATA TYPES	56
<i>Data Types</i>	56
SECTION 2: JAVAPOS INTERFACE DESCRIPTIONS.....	57
JAVAPOS COMMON PROPERTIES, METHODS, AND EVENTS	58
<i>Common Properties</i>	58
<i>JavaPOS Class Names</i>	60
<i>Properties</i>	61
<i>Methods</i>	70
<i>Events</i>	77
PERIPHERAL INTERFACES	82
JAVAPOS: CASH DRAWER	83
<i>Java Command Examples</i>	83
<i>Initializing Properties, Methods, and Events</i>	83
<i>Capabilities, Assignments and Descriptions Properties, Methods, & Events</i>	83
<i>Cash Drawer Operations Properties, Methods, and Events</i>	84
<i>Cash Drawer Terminating Methods</i>	84
JavaPOS: MICR.....	85
<i>Java Command Examples</i>	85
<i>Initializing Properties, Methods, and Events</i>	85
<i>Capabilities, Assignments and Descriptions Properties, Methods, & Events</i>	85

<i>MICR Operations Properties, Methods, and Events</i>	87
<i>MICR Terminating Methods</i>	87
SECTION 3: TECHNICAL DETAILS	88
<i>OPOS to JavaPOS - API Mapping Rules</i>	88
Data Types	88
Property and Method Names	89
Events	89
Constants	89
<i>API Deviations</i>	90
<i>Mapping of CharSet</i>	91
SECTION 4: JAVAPOS CHANGE HISTORY	92
<i>Release 1.3</i>	92
<i>Release 1.4</i>	93
<i>Release 1.5</i>	94
<i>Release 1.6</i>	96
<i>Release 1.7</i>	98
APPENDIX C	
CHANGE HISTORY	1
RELEASE VERSION 1.4	1
RELEASE VERSION 1.5	1
RELEASE VERSION 1.6	3
RELEASE VERSION 1.7	5
RELEASE VERSION 1.8	11
APPENDIX D	
ADDITIONAL SOFTWARE REFERENCES	1
UML REFERENCES	1
<i>Web Location References</i>	1
<i>Reading Material References</i>	1
APPENDIX E	
ADDITIONAL HARDWARE REFERENCES	1
USB PLUSPOWER CONNECTOR	1
<i>Overview</i>	1
<i>Host Side Connector</i>	1
<i>Cable</i>	2
<i>Peripheral Side Connection</i>	2
<i>Web Location References - USB connector EIA approval</i>	2
<i>Reading Material References</i>	3
<i>ARTS Standard Endorsement</i>	3

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 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.
- Operational characteristics and details for implementations which are compliant to the UnifiedPOS architecture. These were added in Appendices A and B for UnifiedPOS Version 1.6.

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 OLE for Retail POS (OPOS) and Java for Retail POS (JavaPOS) committees).

Since the release of UnifiedPOS Version 1.4, the retail standards committees have been maintaining three separate standard documents, UnifiedPOS, JavaPOS and OPOS. The architecture and device characteristics are identical in each of these documents. The addition of new device categories and/or enhancements to existing chapters requires consultation and agreement on the technical content for the standard. However, in addition to that technical work, there is a heavy administrative burden in generating the correct documentation for three different versions of the specification. The current documentation situation is inherently error prone in that the same changes have to be maintained in multiple documents. Confusion is generated in cases where differences have inadvertently appeared in the documentation. In order to simplify the process and bring a higher quality of review to ongoing modifications of the documentation, the standard committee is releasing a consolidated UnifiedPOS specification. Beginning with UnifiedPOS Version 1.6, this specification includes the description of all device categories plus the minor delta information for each of the specific existing implementations, currently JavaPOS and OPOS.

Appendix A includes the definition, goals and deliverables for OPOS. There are explanations for the input/output and device sharing for Microsoft's COM model for the operation of the interface. Event and error handling unique to this implementation are described.

Appendix B includes the definition, goals and deliverables for JavaPOS. There are explanations for the input/output and device sharing for the Java model for the operation of the interface. Event and error handling unique to this implementation are described.

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 implementation references. (See Appendices A & B.)
- 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 formalizes and documents the underlying retail device architecture, 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 and OPOS standards have been established as conformant platform mappings of the UnifiedPOS specification. In UnifiedPOS Version 1.6, appendices were added in order to document specific implementation details for each of these platforms. JavaPOS will be recognized as the only UnifiedPOS conformant, operating system neutral, Java language mapping (See Appendix B). OPOS will be recognized as the only UnifiedPOS conformant language neutral COM mapping (See Appendix A). Future UnifiedPOS mappings to platforms other than Java and COM will be included as appendices to the UnifiedPOS specification as they become available.

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 standards. The content of the UnifiedPOS specification, however, along with the appropriate Appendix, will constitute the definition of how an application can be developed to meet the UnifiedPOS standard. The standards committees do not intend to release future versions of the specific OPOS and JavaPOS documents after the Version 1.6 specification.

Who Should Read This Document

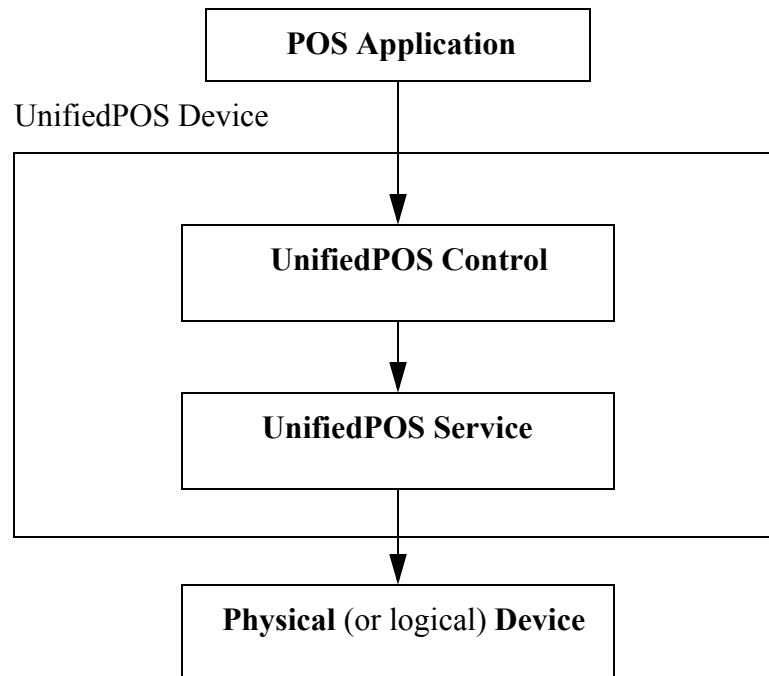
The UnifiedPOS Architecture is targeted to the standard committees that will provide the language specific mapping and Programmer's Guides. However, the application developer who will use POS devices, the system developer who will write POS device code, and the suppliers of POS devices for retail may 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 D-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 14.
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 14.
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 Support data type.

a. Six decimal place precision is required for all computations 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 may need to register for events. The mechanism to do this is implementation specific.

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 9.

*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 }

¹. Not 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

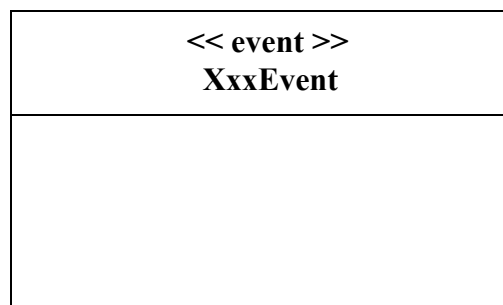
property-string is a property string to apply to the element. For methods an additional property string called 'raises-exception' is defined which means that this method can throw the exception if the implementation language supports exception; otherwise, some mechanism is used to notify the application 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 14 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 22.	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 18.) 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 18.)

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 11 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 error 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

Updated in Release 1.7

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 in program memory, for delivery to the Physical Device as soon as the Physical Device can receive and process it.
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

Updated in Release 1.8.

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 now managed by the POSPower device category, see page 601.

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.

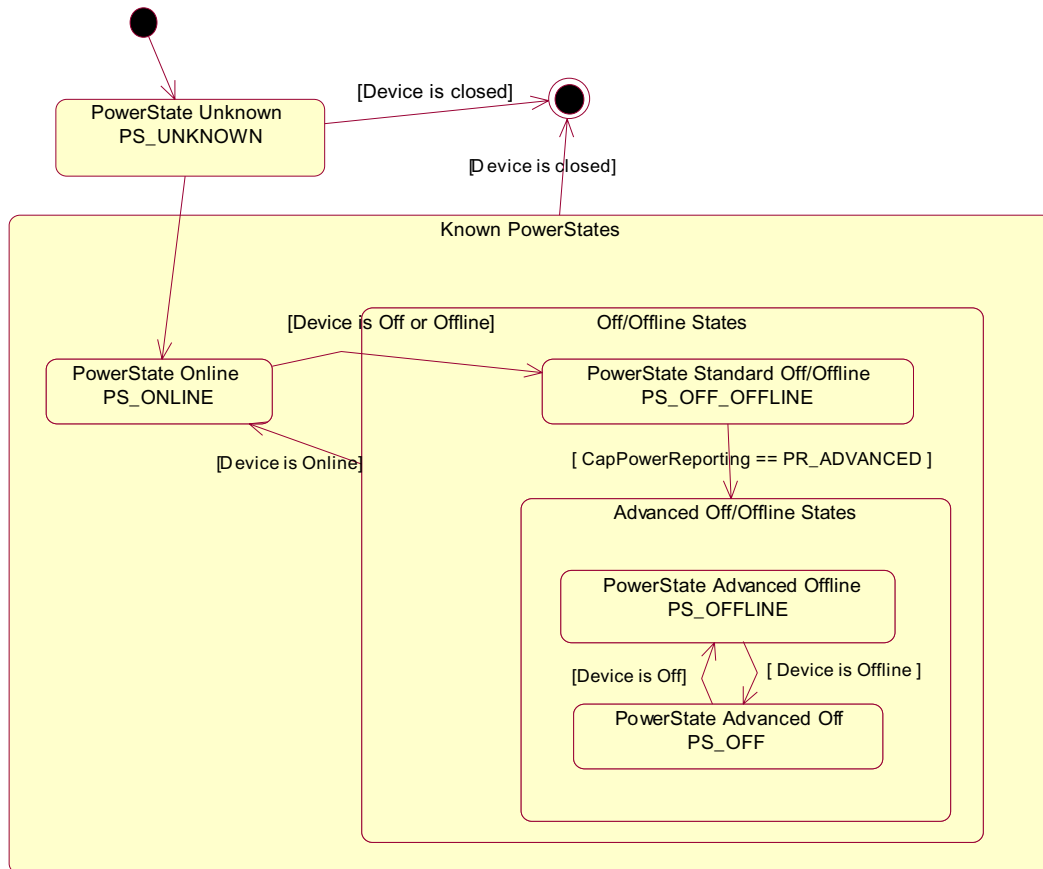
Power reporting only occurs while the device is open, claimed (if the device is exclusive-use), and enabled.

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 Information Reporting Model *Added in Release 1.8.*

POS Applications, as well as System Management agents, frequently need to monitor the current configuration and usage metrics of the various POS devices that are attached to the POS terminal.

Examples of configuration data are the device's *Serial Number*, *Firmware Version*, and *Connection Type*. Examples of usage data for the POSPrinter device are the *Number of Lines Printed*, *Number of Hours Running*, *Number of paper cuts*, etc. Examples of usage data for the Scanner device are the *Number of scans*, *Number of Hours Running*, etc. Examples of usage data for the MSR device are the *Number of successful swipes*, *Number of swipes resulting in errors*, *Number of Hours Running*, etc. See below for examples of XML definitions of the device statistics accumulated per POS device category.

In some cases, the data may be accumulated and stored within the device itself. In other cases, the data may be accumulated by the Service and stored, possibly on the POS terminal or store controller.

In order for multiple applications (for example a POS application and a System Management application) to obtain statistics from the same device, proper care must be taken by both applications so that the device can be made accessible when required. This is done by using the **claim** method and by setting **DeviceEnabled** to true when access to a device is required and then setting **DeviceEnabled** to false and using the **release** method when access to the device is no longer needed. Coordination of device access via this mechanism is the responsibility of the applications themselves.

Statistics Reporting Properties and Methods

The UnifiedPOS device information reporting model adds the following common properties and methods across all device classes.

- **CapStatisticsReporting** property. Identifies the reporting capabilities of the device. When **CapStatisticsReporting** is false, then no statistical data regarding the device is available. This is equivalent to Services compatible with prior versions of the specification. When **CapStatisticsReporting** is true, then some statistical data for the device is available.
- **CapUpdateStatistics** property. Defines whether gathered statistics (or some of them) can be reset/updated by the application. This property is only valid if **CapStatisticsReporting** is true. When **CapUpdateStatistics** is false, then none of the statistical data can be reset/updated by the application. Otherwise, when **CapUpdateStatistics** is true, then (some of) the statistical data can be reset/updated by the application.
- **resetStatistics** method. Can only be called if both **CapStatisticsReporting** and **CapUpdateStatistics** are true. This method resets one, some, or all of the resettable device statistics to zero.
- **retrieveStatistics** method. Can only be called if **CapStatisticsReporting** is true. This method retrieves one, some, or all of the accumulated statistics for the device.
- **updateStatistics** method. Can only be called if both **CapStatisticsReporting** and **CapUpdateStatistics** are true. This method updates one, some, or all of the resettable device statistics to the supplied values.

XML definitions for POS Device Statistics

The XML files containing the UnifiedPOS defined statistics for each device category are provided as downloads from the web sites that also host this specification. These statistics can be referenced individually by name or as a group using the “U_” string as (part of) the parameter to the statistics methods.

Manufacturers/Service providers can add their specific statistics in the provided “ManufacturerSpecific” section. These statistics can be referenced individually by name or as a group using the “M_” string as (part of) the parameter to the statistics methods.

The following table contains the definitions of the information contained in the UnifiedPOS defined DeviceInformation section covering all device categories.

<DeviceInformation> XML Definition Name	Definition description
UnifiedPOSVersion	Version of the UnifiedPOS specification supported
DeviceCategory	Device category (e.g., POSPrinter)
Manufacturer	Device manufacturer’s name
ModelName	Device model name
SerialNumber	Device serial number
ManufactureDate	Device manufacture date
MechanicalRevision	Device hardware revision
FirmwareRevision	Device firmware revision
Interface	Device hardware interface (e.g., serial, USB)

The following is an example of the XML file that describes the “UnifiedPOS” defined statistics for the CashDrawer device category.

```
<?xml version='1.0' ?>
<UPOSStat version="1.8.0" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xmlns="http://www.nrf-arts.org/IXRetail/
namespace/" xsi:schemaLocation="http://www.nrf-arts.org/
IXRetail/namespace/ UPOSStat.xsd">
  <Event>
    <Parameter>
      <Name>NumDrawerGoodOpens</Name>
      <Value>1353</Value>
    </Parameter>
    <Parameter>
      <Name>NumDrawerFailedOpens</Name>
      <Value>2</Value>
    </Parameter>
    <ManufacturerSpecific>
      <Name>MyPersonalStat</Name>
      <Value>14.32</Value>
      <unitofmeasure>meters</unitofmeasure>
    </ManufacturerSpecific>
  </Event>
  <Equipment>
    <Manufacturer>
      <Name>Cashdrawers R Us</Name>
    </Manufacturer>
    <ModelNumber>CD-123</ModelNumber>
    <InstallDate>2000-03-01</InstallDate>
    <SerialNumber>12345</SerialNumber>
    <SensorID UPOS="CashDrawer"/>
    <Software softwaretype="Firmware">
      <Version>1.0</Version>
      <Revision>B</Revision>
    </Software>
    <Interface>RS232</Interface>
  </Equipment>
</UPOSStat>
```

The most up-to-date files defining the XML tag names and example schemas for the statistics for all device categories can be downloaded from the NRF-ARTS web site at <http://www.nrf-arts.org>.

Device States

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

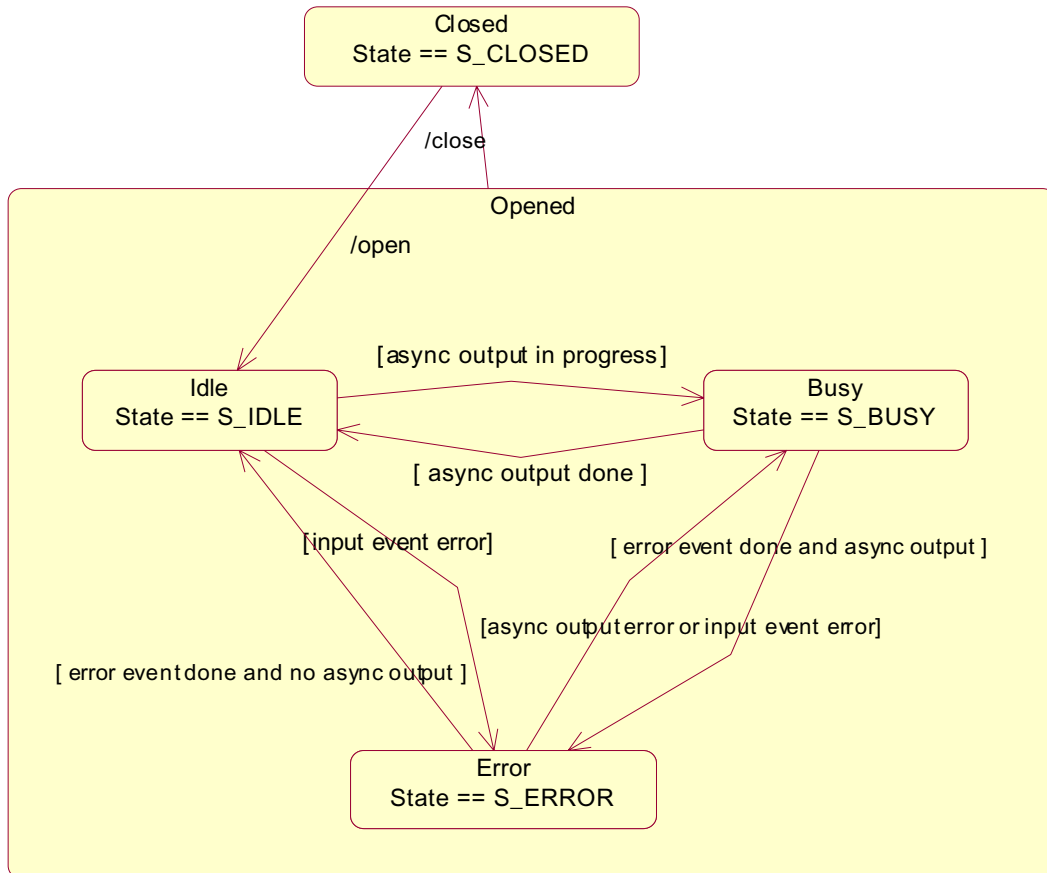
S_CLOSED
S_IDLE
S_BUSY
S_ERROR

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 41) and **DeviceServiceVersion** (see page 43).

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 10.

Summary

Updated in Release 1.8

The following property list is a summary of the JavaPOS Common Properties. This list is used throughout the main UnifiedPOS chapters. Further details may be found in Appendix B, “Common Properties” on page B-58.

The OPOS implementation adds the following Common Properties:

BinaryConversion, OpenResult, ResultCode, and ResultCodeExtended.

Also, the last six properties are replaced by:

ControlObjectDescription, ControlObjectVersion, ServiceObjectDescription, ServiceObjectVersion, DeviceDescription, and DeviceName.

Further details may be found in Appendix A, “Common Properties” on page A-24.

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 }	1.2	1
CapPowerReporting:	<i>int32</i>	{ read-only }	1.3	
CapStatisticsReporting:	<i>boolean</i>	{ read-only }	1.8	
CapUpdateStatistics:	<i>boolean</i>	{ read-only }	1.8	
CheckHealthText:	<i>string</i>	{ read-only }	1.0	
Claimed:	<i>boolean</i>	{ read-only }	1.0	
DataCount:	<i>int32</i>	{ read-only }	1.2	1
DataEventEnabled:	<i>boolean</i>	{ read-write }	1.0	1
DeviceEnabled:	<i>boolean</i>	{ read-write }	1.0	
FreezeEvents:	<i>boolean</i>	{ read-write }	1.0	
OutputID:	<i>int32</i>	{ read-only }	1.0	2
PowerNotify:	<i>int32</i>	{ read-write }	1.3	
PowerState:	<i>int32</i>	{ read-only }	1.3	
State:	<i>int32</i>	{ read-only }	1.0	
DeviceControlDescription:	<i>string</i>	{ read-only }	1.0	
DeviceControlVersion:	<i>int32</i>	{ read-only }	1.0	
DeviceServiceDescription:	<i>string</i>	{ read-only }	1.0	
DeviceServiceVersion:	<i>int32</i>	{ read-only }	1.0	
PhysicalDeviceDescription:	<i>string</i>	{ read-only }	1.0	
PhysicalDeviceName:	<i>string</i>	{ read-only }	1.0	

Usage Notes:

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

Methods (UML operations)

<i>Name</i>	<i>Version</i>
open (logicalDeviceName: <i>string</i>): void { raises-exception }	1.0
close (): void { raises-exception }	1.0
claim ^a (timeout: <i>int32</i>): void { raises-exception }	1.0
release ^a (): void { raises-exception }	1.0
checkHealth (level: <i>int32</i>): void { raises-exception }	1.0
clearInput (): void { raises-exception }	1.0
clearOutput (): void { raises-exception }	1.0
directIO (command: <i>int32</i> , inout data: <i>int32</i> , inout obj: <i>object</i>): void { raises-exception }	1.0
resetStatistics (statisticsBuffer: <i>string</i>): void { raises-exception }	1.8
retrieveStatistics (inout statisticsBuffer: <i>string</i>): void { raises-exception }	1.8
updateStatistics (statisticsBuffer: <i>string</i>): void { raises-exception }	1.8

- a. **Note:** In the OPOS environment starting with Release 1.5, the **Claim** and **Release** methods are also defined as **ClaimDevice** and **ReleaseDevice** respectively 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>Version</i>	<i>Usage Notes</i>
upos::events::DataEvent			1.0	1
Status:	<i>int32</i>	{ read-only }		
upos::events::DirectIOEvent			1.0	
EventNumber:	<i>int32</i>	{ read-only }		
Data:	<i>int32</i>	{ read-write }		
Obj:	<i>object</i>	{ read-write }		
upos::events::ErrorEvent			1.0	
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			1.0	2
OutputID:	<i>int32</i>	{ read-only }		
upos::events::StatusUpdateEvent			1.0	
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.

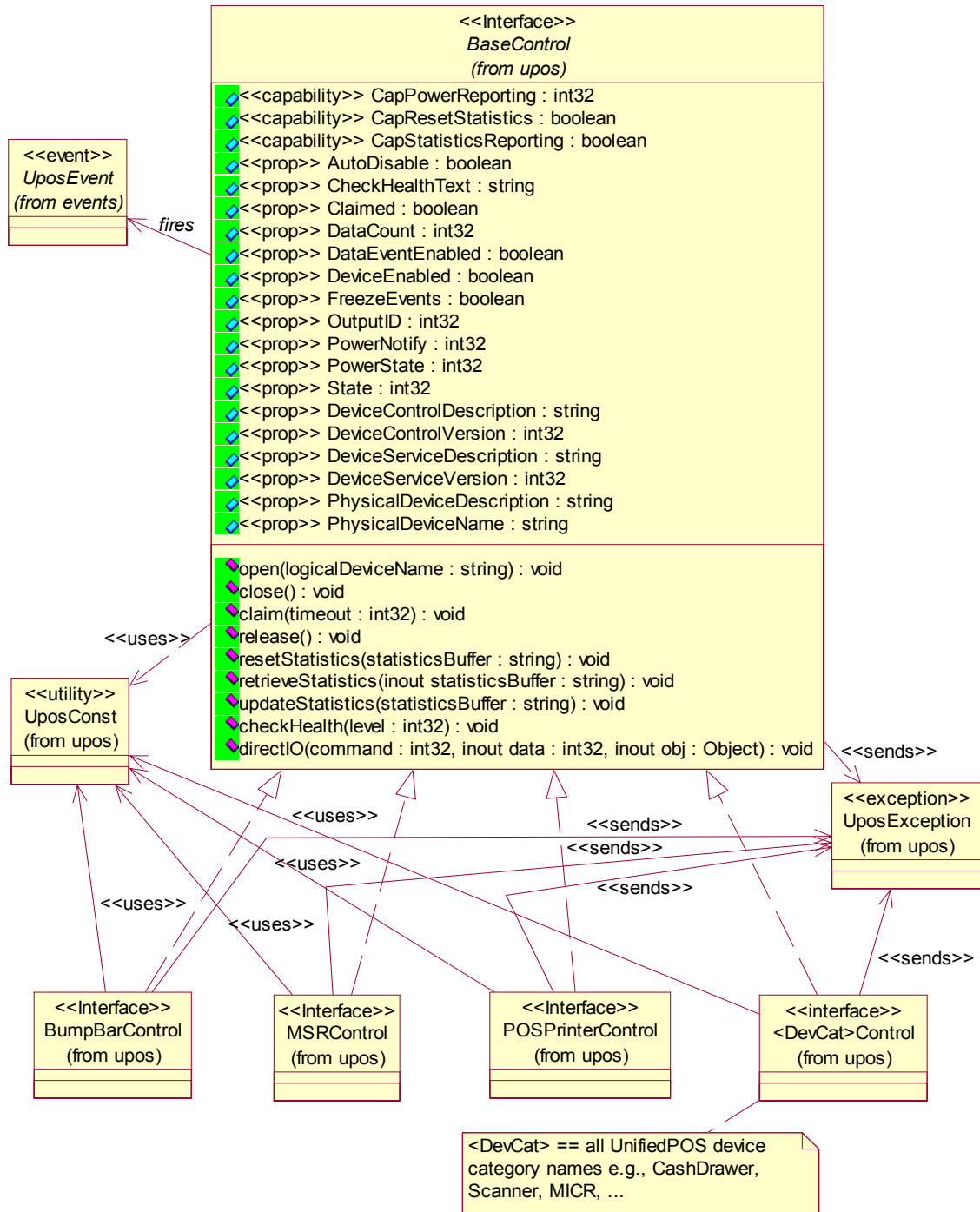
General Information

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 10 for additional information.

The following diagram shows the relationships between the Common classes.



Notes: **AutoDisable**, **DataCount**, and **DataEventEnabled** are used only with Devices that have Event Driven Input.
OutputID is used only with Asynchronous Output Devices.

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 16.
See Also	“Device Input Model” on page 18.

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> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <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> </tbody> </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 22, PowerState Property, PowerNotify Property.								

CapStatisticsReporting Property**Added in Release 1.8**

Syntax	CapStatisticsReporting: <i>boolean</i> { read-only }
Remarks	If true, the device accumulates and can provide various statistics regarding usage; otherwise no usage statistics are accumulated. The information accumulated and reported is device specific, and is retrieved using the retrieveStatistics method. 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 16.
See Also	retrieveStatistics Method.

CapUpdateStatistics Property**Added in Release 1.8**

Syntax	CapUpdateStatistics: <i>boolean</i> { read-only }
Remarks	If true, the device statistics, or some of the statistics, can be reset to zero using the resetStatistics method, or updated using the updateStatistics method. If CapStatisticsReporting is false, then CapUpdateStatistics is also 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 16.
See Also	CapStatisticsReporting Property, resetStatistics Method, updateStatistics Method.

CheckHealthText Property

Syntax	CheckHealthText: <i>string</i> { read-only }
Remarks	Holds the results of the most recent call to the checkHealth method. The following examples illustrate some possible diagnoses: <ul style="list-style-type: none">• “Internal HCheck: Successful”• “External HCheck: Not Responding”• “Interactive HCheck: Complete” This property is empty (“”) before the first call to the checkHealth method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16
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 16.
See Also	“Device Initialization and Finalization” on page 12, “Device Sharing Model” on page 14, 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 16.
See Also	“Device Input Model” on page 18, 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 16.
See Also	“Events” on page 15, 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: <i>int32</i> { read-only }								
Remarks	<p>Holds the UnifiedPOS Control version number.</p> <p>Three version levels are specified, as follows:</p> <table border="1"> <thead> <tr> <th>Version Level</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>Major</td> <td> <p>The “millions” place.</p> <p>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.</p> </td> </tr> <tr> <td>Minor</td> <td> <p>The “thousands” place.</p> <p>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.</p> </td> </tr> <tr> <td>Build</td> <td> <p>The “units” place.</p> <p>Internal level provided by the UnifiedPOS Control developer. Updated when corrections are made to the UnifiedPOS Control implementation.</p> </td> </tr> </tbody> </table> <p>A sample version number is:</p> <pre> 1002038 </pre> <p>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.</p> <p>This property is always readable.</p>	Version Level	Description	Major	<p>The “millions” place.</p> <p>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.</p>	Minor	<p>The “thousands” place.</p> <p>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.</p>	Build	<p>The “units” place.</p> <p>Internal level provided by the UnifiedPOS Control developer. Updated when corrections are made to the UnifiedPOS Control implementation.</p>
Version Level	Description								
Major	<p>The “millions” place.</p> <p>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.</p>								
Minor	<p>The “thousands” place.</p> <p>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.</p>								
Build	<p>The “units” place.</p> <p>Internal level provided by the UnifiedPOS Control developer. Updated when corrections are made to the UnifiedPOS Control implementation.</p>								
Errors	None.								
See Also	“Version Handling” on page 31, 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 22 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 16.
See Also	"Device Initialization and Finalization" on page 12.

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 16.

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 16.

See Also “Version Handling” on page 31, **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 16.

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 16.
See Also	“Device Output Models” on page 21, 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 to PN_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 16.

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

Value	Meaning
E_ILLEGAL	One of the following occurred: <ul style="list-style-type: none"> The device is already enabled. PowerNotify = PN_ENABLED but CapPowerReporting = PR_NONE.

See Also “Device Power Reporting Model” on page 22, **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 22, **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 16.
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 16.
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 Information Reporting Model” on page 26.

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 16.

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: *int32*):**
 void { raises-exception }

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

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

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

Errors A UposException may be thrown when this method is invoked. For further information, “Errors” on page 16.

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

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 14, **release** Method.

clearInput Method

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

Remarks Clears all device input that has been buffered.

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

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

See Also “Device Input Model” on page 18.

clearOutput Method**Updated in Release 1.7**

- Syntax** **clearOutput ():**
 void { raises-exception }
- Remarks** Clears all buffered output data, including all asynchronous output. Also, when possible, halts outputs that are in progress.
Any output error events that are enqueued – usually waiting for **FreezeEvents** to be set to false – are also cleared.
- Errors** A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.
- See Also** “Device Output Models” on page 21.

close Method

- Syntax** **close ():**
 void { raises-exception }
- Remarks** Releases the device and its resources.
If the **DeviceEnabled** property is true, then the device is disabled.
If the **Claimed** property is true, then exclusive access to the device is released.
- Errors** A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.
- See Also** “Device Initialization and Finalization” on page 12, **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 16.
- See Also** **DirectIOEvent**.

open Method**Updated in Release 1.7**

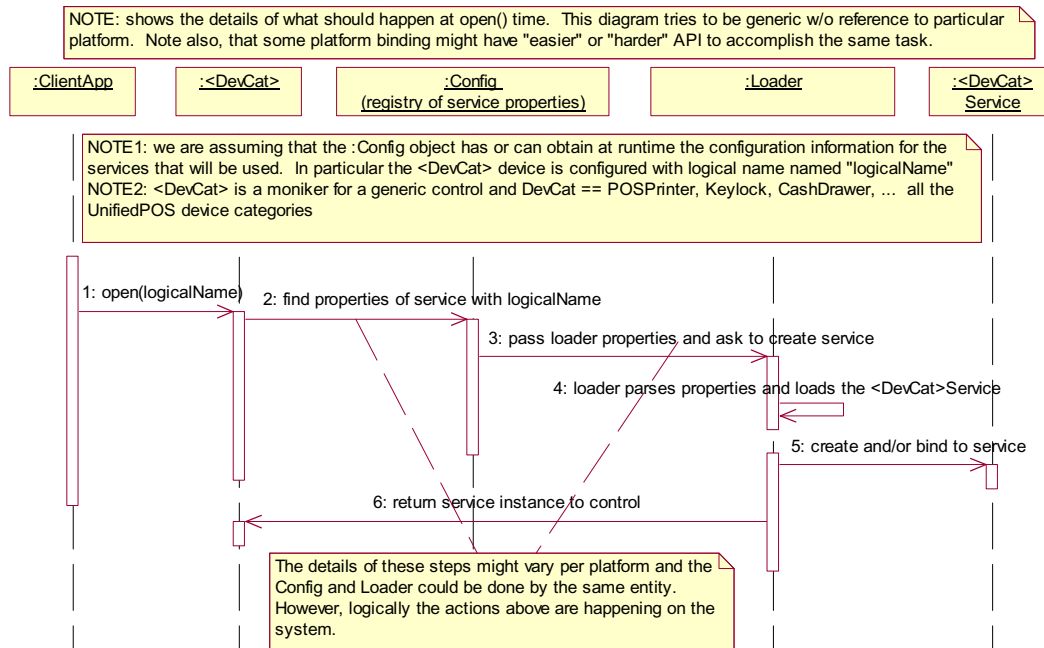
Syntax **open (logicalDeviceName: *string*):**
 void { raises-exception }

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 the JavaPOS Configurator/Loader (JCL) or the 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.

The following sequence diagram shows the details of what needs to happen during the **open** method call processing to allow the creation of the Service and its binding to the Control.



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 16.

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 12, “Version Handling” on page 31, **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 16.

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 14, **claim** Method.

resetStatistics Method**Added in Release 1.8**

Syntax `resetStatistics (statisticsBuffer: string):
 void { raises-exception }`

Parameter	Description
<i>statisticsBuffer</i>	The data buffer defining the statistics that are to be reset.

This is a comma-separated list of name(s), where an empty string (“”) means ALL resettable statistics are to be reset, “U_” means all UnifiedPOS defined resettable statistics are to be reset, “M_” means all manufacturer defined resettable statistics are to be reset, and “actual_name1, actual_name2” (from the XML file definitions) means that the specifically defined resettable statistic(s) are to be reset.

Remarks Resets the defined resettable statistics in a device to zero. All the requested statistics must be successfully reset in order for this method to complete successfully, otherwise an *ErrorCode* of E_EXTENDED is returned.

Both **CapStatisticsReporting** and **CapUpdateStatistics** must be true in order to successfully use this method.

This method is always executed synchronously.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	CapStatisticsReporting or CapUpdateStatistics is false, or the named statistic is not defined/resettable.
E_EXTENDED	<i>ErrorCodeExtended</i> = ESTATS_ERROR: At least one of the specified statistics could not be reset.

See Also **CapStatisticsReporting** Property, **CapUpdateStatistics** Property.

retrieveStatistics Method**Added in Release 1.8**

Syntax `retrieveStatistics (inout statisticsBuffer: string):
 void { raises-exception }`

Parameter	Description
<i>statisticsBuffer</i>	The data buffer defining the statistics to be retrieved and in which the retrieved statistics are placed.

This is a comma-separated list of name(s), where an empty string (“”) means ALL statistics are to be retrieved, “U_” means all UnifiedPOS defined statistics are to be retrieved, “M_” means all manufacturer defined statistics are to be retrieved, and “actual_name1, actual_name2” (from the XML file definitions) means that the specifically defined statistic(s) are to be retrieved.

Remarks Retrieves the requested statistics from a device.

CapStatisticsReporting must be true in order to successfully use this method.

This method is always executed synchronously.

All calls to **retrieveStatistics** will return the following XML as a minimum:

```
<?xml version='1.0'?>
<UPOSStat version="1.8.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://www.nrf-arts.org/IXRetail/namespace/"
  xsi:schemaLocation=
    "http://www.nrf-arts.org/IXRetail/namespace/
    UPOSStat.xsd">
  <Event>
    <Parameter>
      <Name>RequestedStatistic</Name>
      <Value>1234</Value>
    </Parameter>
  </Event>
  <Equipment>
    <Manufacturer>
      <Name>Device Manufacturer</Name>
    </Manufacturer>
    <ModelNumber>Device Model Number</ModelNumber>
    <SensorID UPOS="POSPrinter"/>
  </Equipment>
</UPOSStat>
```

If the application requests a statistic name that the device does not support, the `<Parameter>` entry will be returned with an empty `<Value>`. e.g.,

```
<Parameter>
  <Name>RequestedStatistic</Name>
  <Value></Value>
</Parameter>
```

All statistics that the device collects that are manufacturer specific (not defined in the schema) will be returned in a `<ManufacturerSpecific>` tag instead of a `<Parameter>` tag. e.g.,

```
<ManufacturerSpecific>
  <Name>TheAnswer</Name>
  <Value>42</Value>
</ManufacturerSpecific>
```

When an application requests all statistics from the device, the device will return a `<Parameter>` entry for every defined statistic for the device category as defined by the XML schema version specified by the version attribute in the `<UPOSStat>` tag. If the device does not record any of the statistics, the `<Value>` tag will be empty.

The most up-to-date files defining the XML tag names and example schemas for the statistics for all device categories can be downloaded from the NRF-ARTS web site at <http://www.nrf-arts.org>.

Errors A `UpoxException` may be thrown when this method is invoked. For further information, see "Errors" on page 16.

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

Value	Meaning
E_ILLEGAL	CapStatisticsReporting is false or the named statistic is not defined.

See Also **CapStatisticsReporting** Property.

updateStatistics Method**Added in Release 1.8**

Syntax **updateStatistics (statisticsBuffer: *string*):**
 void { raises-exception }

Parameter	Description
<i>statisticsBuffer</i>	The data buffer defining the statistics with values that are to be updated.

This is a comma-separated list of name-value pair(s), where an empty string name (“”=value1”) means ALL resettable statistics are to be set to the value “value1”, “U_=value2” means all UnifiedPOS defined resettable statistics are to be set to the value “value2”, “M_=value3” means all manufacturer defined resettable statistics are to be set to the value “value3”, and “actual_name1=value4, actual_name2=value5” (from the XML file definitions) means that the specifically defined resettable statistic(s) are to be set to the specified value(s).

Remarks Updates the defined resettable statistics in a device. All the requested statistics must be successfully updated in order for this method to complete successfully, otherwise an *ErrorCode* of E_EXTENDED is returned.

Both **CapStatisticsReporting** and **CapUpdateStatistics** must be true in order to successfully use this method.

This method is always executed synchronously.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

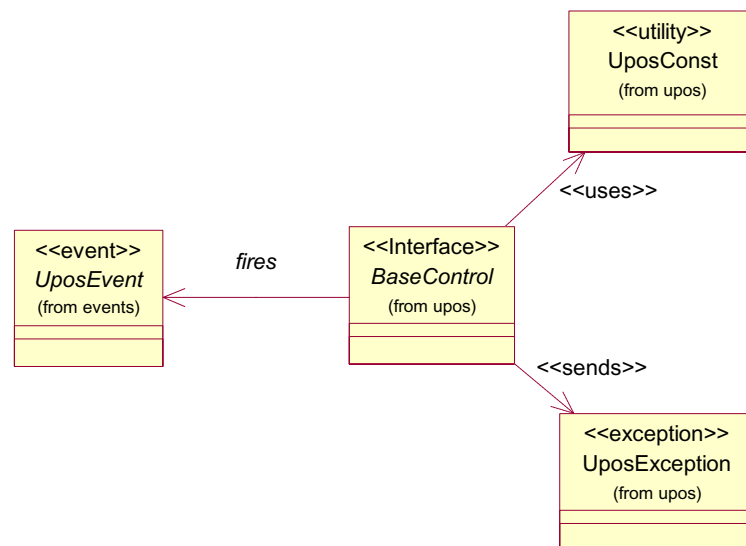
Value	Meaning
E_ILLEGAL	CapStatisticsReporting or CapUpdateStatistics is false, or the named statistic is not defined/updatable.
E_EXTENDED	<i>ErrorCodeExtended</i> = ESTATS_ERROR: At least one of the specified statistics could not be updated.

See Also **CapStatisticsReporting** Property, **CapUpdateStatistics** Property.

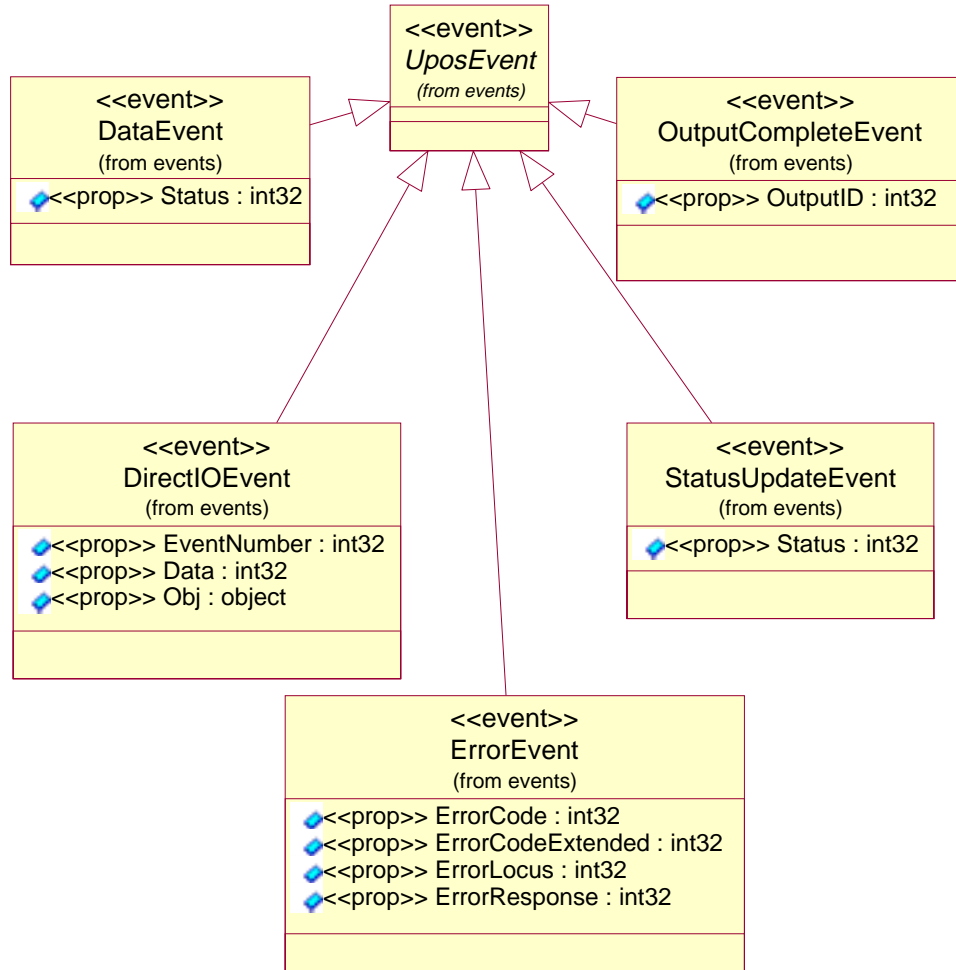
Events (UML interfaces)

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 15, “Device Input Model” on page 18, **DataEventEnabled** Property, **FreezeEvents** Property.

DirectIOEvent**Updated in Release 1.7**

```

<<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. ¹

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 15, **directIO** Method.

¹. In the OPOS environment, the format of this data depends upon the value of the **BinaryConversion** property. See **BinaryConversion** property on page A-28.

ErrorEvent**Updated in Release 1.7**

```

<<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 16.
<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
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 application’s error event handler can set the *ErrorResponse* attribute to one of the following values:

Value	Meaning
ER_RETRY	Retry the input or asynchronous output. The error state is exited. May be valid only when locus is EL_INPUT. Default when locus is EL_OUTPUT.
ER_CLEAR	Clear all buffered output data (including all asynchronous output) or buffered input data. The error state is exited. Default when locus is EL_INPUT.

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 EL_INPUT.

Use only when locus is EL_INPUT_DATA. Default when locus is 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 18, "Device Information Reporting Model" on page 26.

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 21.

StatusUpdateEvent

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

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

Attribute This event contains the following attribute:

Attribute	Type	Description
<i>Status</i>	<i>int32</i>	Device category-specific status, describing the type of status change.

Power State Reporting adds additional *Status* values of:

Value	Meaning
SUE_POWER_ONLINE	The device is powered on and ready for use. Can be returned if CapPowerReporting = PR_STANDARD or PR_ADVANCED.
SUE_POWER_OFF	The device is off or detached from the terminal. Can only be returned if CapPowerReporting = PR_ADVANCED.
SUE_POWER_OFFLINE	The device is powered on but is either not ready or not able to respond to requests. Can only be returned if CapPowerReporting = PR_ADVANCED.
SUE_POWER_OFF_OFFLINE	The device is either off or offline. Can only be returned if CapPowerReporting = PR_STANDARD.

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

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

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

See Also “Events” on page 15, “Device Power Reporting Model” on page 22, **CapPowerReporting** Property, **PowerNotify** Property.

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</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
CapStatisticsReporting:	<i>boolean</i>	{ read-only }	1.8	open
CapUpdateStatistics:	<i>boolean</i>	{ read-only }	1.8	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

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
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
CapTone:	<i>boolean</i>	{ read-only }	1.3	open, claim, & enable
CurrentUnitID:	<i>int32</i>	{ read-write }	1.3	open, claim, & enable
ErrorString:	<i>string</i>	{ read-only }	1.3	open
ErrorUnits:	<i>int32</i>	{ read-only }	1.3	open
EventString:	<i>string</i>	{ read-only }	1.3	open & claim
EventUnitID:	<i>int32</i>	{ read-only }	1.3	open & claim
EventUnits:	<i>int32</i>	{ read-only }	1.3	open & claim
Keys:	<i>int32</i>	{ read-only }	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

Methods (UML operations)**Common**

<i>Name</i>	<i>Version</i>
open (logicalDeviceName: <i>string</i>): void { raises-exception }	1.3
close (): void { raises-exception, use after open }	1.3
claim (timeout: <i>int32</i>): void { raises-exception, use after open }	1.3
release (): void { raises-exception, use after open, claim }	1.3
checkHealth (level: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3
clearInput (): void { raises-exception, use after open, claim }	1.3
clearOutput (): void { raises-exception, use after open, claim }	1.3
directIO (command: <i>int32</i>, inout data: <i>int32</i>, inout obj: <i>object</i>): void { raises-exception, use after open }	1.3
resetStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
retrieveStatistics (inout statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
updateStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8

Specific

<i>Name</i>	
bumpBarSound (units: <i>int32</i>, frequency: <i>int32</i>, duration: <i>int32</i>, numberOfCycles: <i>int32</i>, interSoundWait: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3
setKeyTranslation (units: <i>int32</i>, scanCodes: <i>int32</i>, logicalKey: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>
upos::events::DataEvent			1.3
Status:	<i>int32</i>	{ read-only }	
upos::events::DirectIOEvent			1.3
EventNumber:	<i>int32</i>	{ read-only }	
Data:	<i>int32</i>	{ read-write }	
Obj:	<i>object</i>	{ read-write }	
upos::events::ErrorEvent			1.3
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			1.3
OutputID:	<i>int32</i>	{ read-only }	
upos::events::StatusUpdateEvent			1.3
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.

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 **ErrorString** 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

Updated in Release 1.7

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 in program memory, for delivery to the Physical Device as soon as the Physical Device can receive and process it, 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.

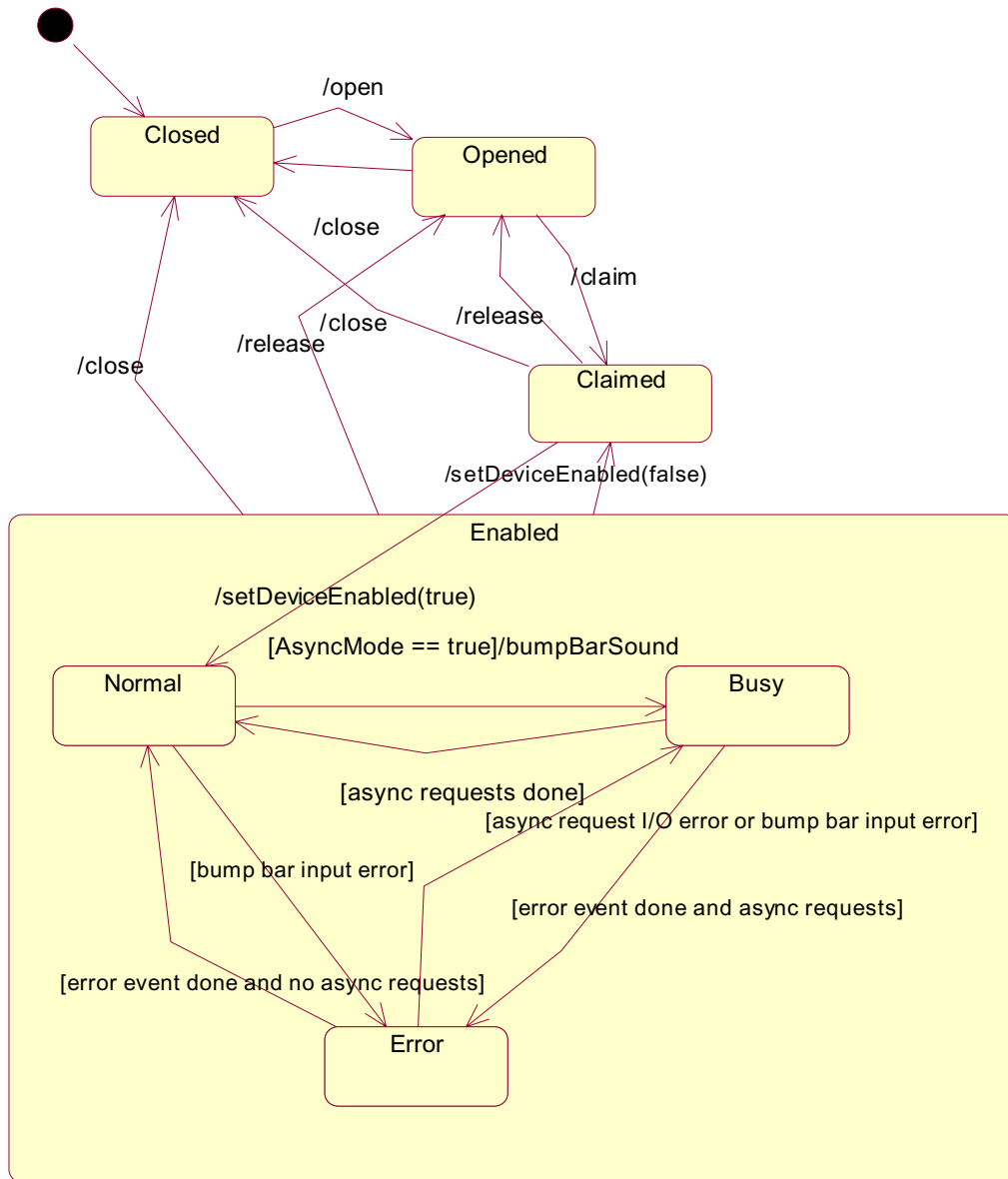
- Asynchronous output is performed on a first-in first-out basis.
- All output buffered may be deleted by setting the **CurrentUnitID** property and calling the **clearOutput** method. An **OutputCompleteEvent** will not be enqueued for cleared output. This method also stops any output that may be in 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	If true, then the bumpBarSound method will be performed asynchronously. If false, tones are generated synchronously. This property is initialized to false by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	bumpBarSound Method, “Device Output Models” on page 21.

AutoToneDuration Property

Syntax	AutoToneDuration: <i>int32</i> { read-write, access after open-claim-enable }
Remarks	Holds the duration (in milliseconds) of the automatic tone for the bump bar unit specified by the CurrentUnitID property. This property is initialized to the default value for each online bump bar unit when the device is first enabled following the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	CurrentUnitID Property.

AutoToneFrequency Property

Syntax	AutoToneFrequency: <i>int32</i> { read-write, access after open-claim-enable }
Remarks	Holds the frequency (in Hertz) of the automatic tone for the bump bar unit specified by the CurrentUnitID property. This property is initialized to the default value for each online bump bar unit when the device is first enabled following the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
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 16.
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 16.
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 16.

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 16.
See Also	BumpBarDataCount Property, DataEvent Event, “Device Input Model” on page 18.

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 16.
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 16.
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 16.
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 16.
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 16.
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 16.
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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
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 BB_UID_1 (bit 0 on) through BB_UID_32 (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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
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 16.

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

Value	Meaning
E_ILLEGAL	<p>One of the following errors occurred:</p> <p><i>numberOfCycles</i> is neither a positive, non-zero value nor FOREVER.</p> <p><i>numberOfCycles</i> is FOREVER when AsyncMode is false.</p> <p>A negative <i>interSoundWait</i> was specified.</p> <p><i>units</i> is zero or a non-existent unit was specified.</p> <p>A unit in <i>units</i> does not support the CapTone capability.</p> <p>The ErrorUnits and ErrorString properties may be updated before the exception is thrown.</p>
E_FAILURE	<p>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.)</p>

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

clearInput Method (Common)

Syntax	clearInput (): void { raises-exception, use after open-claim }
Remarks	Clears the device input that has been buffered for the unit specified by the CurrentUnitID property. Any data events that are enqueued – usually waiting for DataEventEnabled to be set to true and FreezeEvents to be set to false – are also cleared.
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.
See Also	CurrentUnitID Property, “Device Input Model” on page 18.

clearOutput Method (Common)***Updated in Release 1.7***

Syntax	clearOutput (): void { raises-exception, use after open-claim }
Remarks	Clears the tone outputs that have been buffered, including all asynchronous output, for the unit specified by the CurrentUnitID property. Any output complete and output error events that are enqueued – usually waiting for DataEventEnabled to be set to true and FreezeEvents to be set to false – are also cleared.
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.
See Also	CurrentUnitID Property, “Device Output Models” on page 21.

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 18, **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 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 Bump Bar devices which may not have any knowledge of the Service's need for this event.

See Also "Events" on page 15, **directIO** Method.

ErrorEvent**Updated in Release 1.7**

```

<< 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 attributes:

Attributes	Type	Description
<i>ErrorCode</i>	<i>int32</i>	Error code causing the error event. See a list of Error Codes on page 16.
<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 all buffered output data (including all 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 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 21, “Device Information Reporting Model” on page 26, **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 21.

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 StatusUpdateEvent values</i> . See “StatusUpdateEvent” description on page 63.

Remarks Enqueued when the bump bar device detects a power state change.
 Deviation from the standard **StatusUpdateEvent** (See “StatusUpdateEvent” description on page 63)

- 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</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
CapStatisticsReporting:	<i>boolean</i>	{ read-only }	1.8	open
CapUpdateStatistics:	<i>boolean</i>	{ read-only }	1.8	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

Properties (Continued)

<i>Specific</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**

<i>Name</i>	<i>Version</i>
open (logicalDeviceName: <i>string</i>): void { raises-exception }	1.2
close (): void { raises-exception, use after open }	1.2
claim (timeout: <i>int32</i>): void { raises-exception, use after open }	1.2
release (): void { raises-exception, use after open, claim }	1.2
checkHealth (level: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.2
clearInput (): void { raises-exception, use after open, claim }	1.5
clearOutput (): void { }	<i>Not supported</i>
directIO (command: <i>int32</i>, inout data: <i>int32</i>, inout obj: <i>object</i>): void { raises-exception, use after open }	1.2
resetStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
retrieveStatistics (inout statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
updateStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8

Specific

<i>Name</i>	
beginDeposit (): void { raises-exception, use after open, claim, enable }	1.5
dispenseCash (cashCounts: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.2
dispenseChange (amount: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.2
endDeposit (success: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.5
fixDeposit (): void { raises-exception, use after open, claim, enable }	1.5
pauseDeposit (control: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.5
readCashCounts (inout cashCounts: <i>string</i>, inout discrepancy: <i>boolean</i>): void { raises-exception, use after open, claim, enable }	1.2

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>
upos::events::DataEvent			1.5
Status:	<i>int32</i>	{ read-only }	
upos::events::DirectIOEvent			1.2
EventNumber:	<i>int32</i>	{ read-only }	
Data:	<i>int32</i>	{ read-write }	
Obj:	<i>object</i>	{ read-write }	
upos::events::ErrorEvent		<i>Not Supported</i>	
upos::events::OutputCompleteEvent		<i>Not Supported</i>	
upos::events::StatusUpdateEvent			1.2
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:

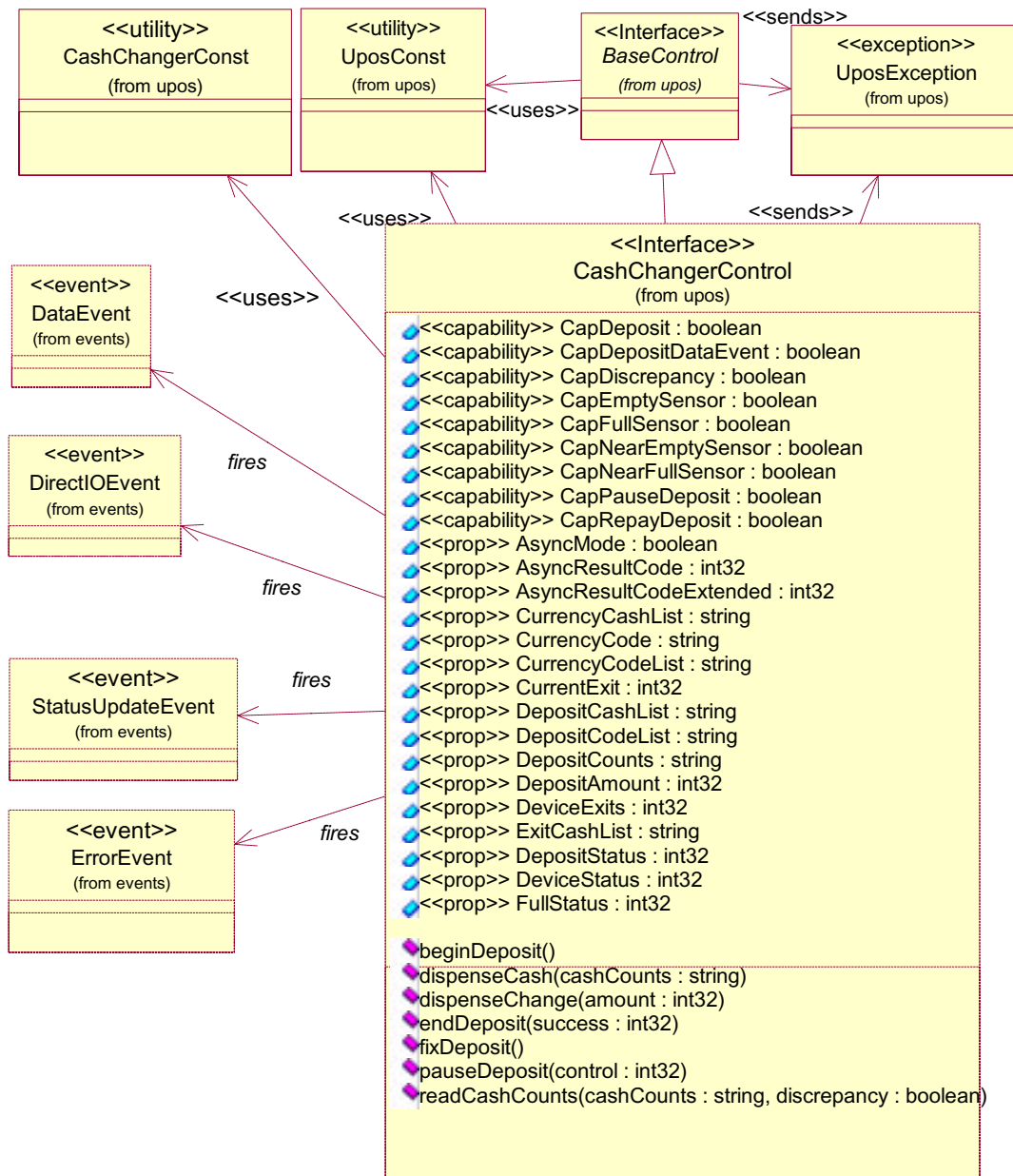
- Reporting the fullness levels of the Cash Changer’s cash units. Conditions which may be indicated include empty, near empty, full, and near full states.
- Reporting of a possible (or probable) cash count discrepancy in the data reported by the **readCashCounts** method.

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 `UposException` 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.

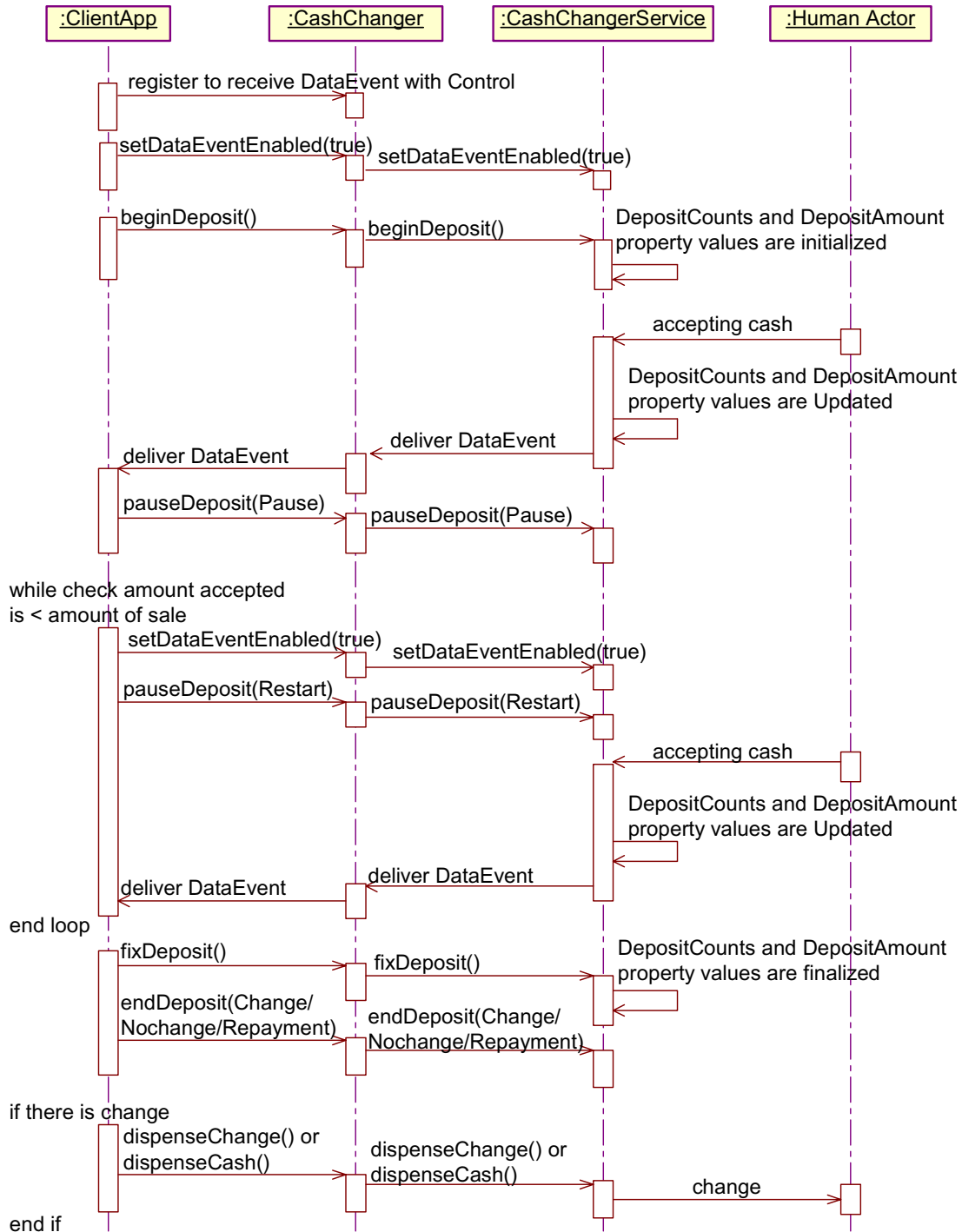
- May support more than one currency. The **CurrencyCode** property may be set to the currency, selecting from a currency in the list **CurrencyCodeList**. **CurrencyCashList**, **ExitCashList**, **dispenseCash**, **dispenseChange** and **readCashCounts** all act upon the current currency only.
- Sets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

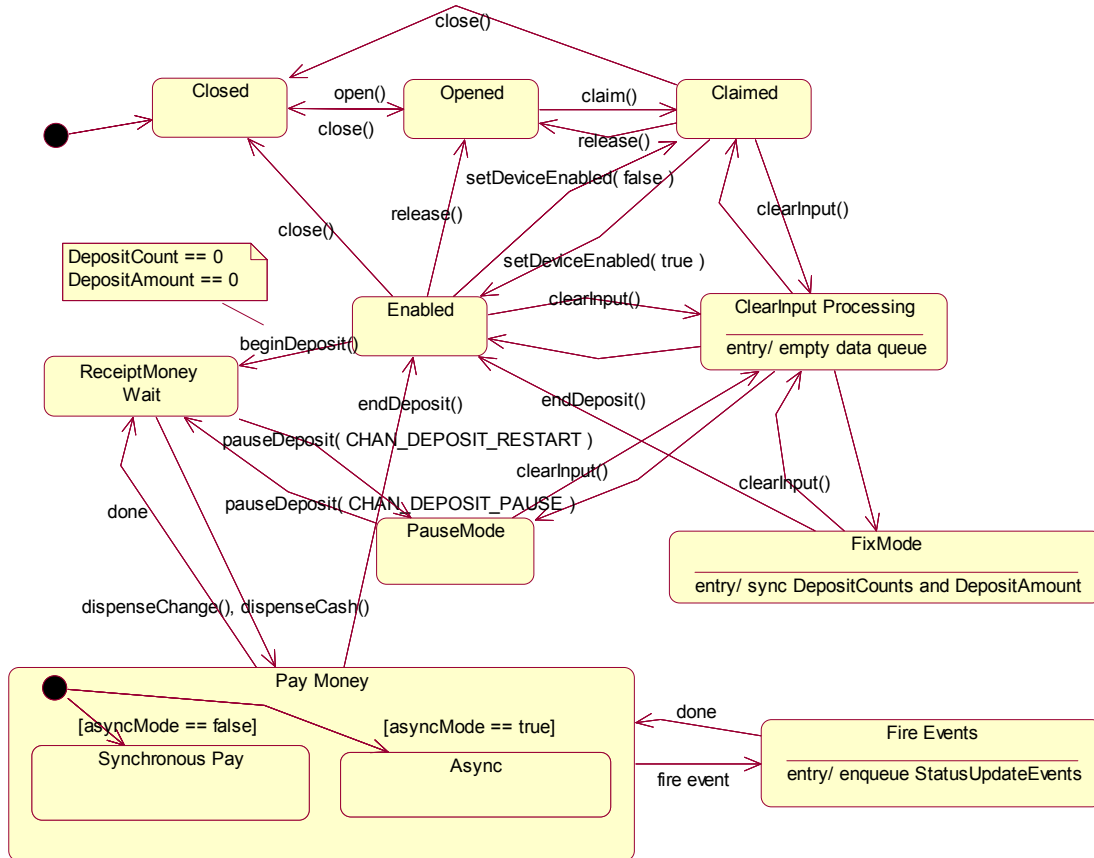
Cash Changer Sequence Diagram

Added in Release 1.7

NOTE: we are assuming that the :ClientApp already successfully open, Claimed and enabled the CashChanger device. This means that the Claimed, DeviceEnabled properties are == true



Cash Changer State Diagram *Updated in Release 1.8*



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 16.
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 16.
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 16.
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 16.
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 16.
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 16.
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 16.
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 16.
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 16.
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 16.
See Also	pauseDeposit Method.

CapRepayDeposit Property

Added in Release 1.5

- Syntax** **CapRepayDeposit:** *boolean* { 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 16.
- See Also** **endDeposit** Method.

CurrencyCashList Property

- Syntax** **CurrencyCashList:** *string* { read-only, access after open }
- Remarks** Holds the cash units supported in the Cash Changer for the currency represented by the **CurrencyCode** Property.
- 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.
- Below are sample **CurrencyCashList** values in Japan.
- “1,5,10,50,100,500” ---
1, 5, 10, 50, 100, 500 yen coin.
 - “1,5,10,50,100,500;1000,5000,10000” ---
1, 5, 10, 50, 100, 500 yen coin and 1000, 5000, 10000 yen bill.
 - “;1000,5000,10000” ---
1000, 5000, 10000 yen bill.
- This property is initialized by the **open** method, and is updated when **CurrencyCode** is set.
- Errors** A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
- See Also** **CurrencyCode** Property.

CurrencyCode Property

- Syntax** **CurrencyCode:** *string* { 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** A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	A value was specified that is not within CurrencyCodeList .

- See Also** **CurrencyCodeList** Property.

CurrencyCodeList Property

- Syntax** **CurrencyCodeList:** *string* { read-only, access after open }
- Remarks** 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.
- 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 16.
- 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 <code>UposException</code> may be thrown when this property is accessed. For further information, see “Errors” on page 16.
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 <code>UposException</code> may be thrown when this property is accessed. For further information, see “Errors” on page 16.
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 16.
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 16.
See Also	CurrencyCode Property.

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

- Syntax** **DepositStatus: *int32* { 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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

DeviceExits Property

- Syntax** **DeviceExits: *int32* { read-only, access after open }**
- Remarks** The number of exits for dispensing cash.
- 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 16.
- 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> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>CHAN_STATUS_OK</td> <td>The current condition of the Cash Changer is satisfactory.</td> </tr> <tr> <td>CHAN_STATUS_EMPTY</td> <td>Some cash slots are empty.</td> </tr> <tr> <td>CHAN_STATUS_NEAREMPTY</td> <td>Some cash slots are nearly empty.</td> </tr> <tr> <td>CHAN_STATUS_JAM</td> <td>A mechanical fault has occurred.</td> </tr> </tbody> </table> <p>This property is initialized and kept current while the device is enabled. If more than one condition is present, then the order of precedence starting at the highest is: 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 16.										

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 16.
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> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>CHAN_STATUS_OK</td> <td>All cash slots are neither nearly full nor full.</td> </tr> <tr> <td>CHAN_STATUS_FULL</td> <td>Some cash slots are full.</td> </tr> <tr> <td>CHAN_STATUS_NEARFULL</td> <td>Some cash slots are nearly full.</td> </tr> </tbody> </table> <p>This property is initialized and kept current while the device is enabled.</p>	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 16.								

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> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>E_ILLEGAL</td> <td>Either the Cash Changer does not support cash acceptance, or the call sequence is not correct.</td> </tr> </tbody> </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.				

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

See Also "Events" on page 15, **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 63.

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

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

Remarks Fired when the Cash Changer detects a status change.
 For changes in the fullness levels, the Cash Changer is only able to fire **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 15.

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</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
CapStatisticsReporting:	<i>boolean</i>	{ read-only }	1.8	open
CapUpdateStatistics:	<i>boolean</i>	{ read-only }	1.8	open
CheckHealthText:	<i>string</i>	{ read-only }	1.0	open
Claimed:	<i>boolean</i>	{ read-only }	1.0	open
DataCount:	<i>int32</i>	{ read-only }	1.2	Not Supported
DataEventEnabled:	<i>boolean</i>	{ read-write }	1.0	Not Supported
DeviceEnabled:	<i>boolean</i>	{ read-write }	1.0	open
FreezeEvents:	<i>boolean</i>	{ read-write }	1.0	open
OutputID:	<i>int32</i>	{ read-only }	1.0	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 }	1.0	--
DeviceControlDescription:	<i>string</i>	{ read-only }	1.0	--
DeviceControlVersion:	<i>int32</i>	{ read-only }	1.0	--
DeviceServiceDescription:	<i>string</i>	{ read-only }	1.0	open
DeviceServiceVersion:	<i>int32</i>	{ read-only }	1.0	open
PhysicalDeviceDescription:	<i>string</i>	{ read-only }	1.0	open
PhysicalDeviceName:	<i>string</i>	{ read-only }	1.0	open

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 }	1.0	open
CapStatusMultiDrawerDetect:	<i>boolean</i>	{ read-only }	1.5	open
DrawerOpened:	<i>boolean</i>	{ read-only }	1.0	open & enable

Methods (UML operations)**Common**

<i>Name</i>	<i>Version</i>
open (logicalDeviceName: <i>string</i>): void { raises-exception }	1.0
close (): void { raises-exception, use after open }	1.0
claim (timeout: <i>int32</i>): void { raises-exception, use after open }	1.0
release (): void { raises-exception, use after open, claim }	1.0
checkHealth (level: <i>int32</i>): void { raises-exception, use after open, enable }	1.0 <i>Note</i>
clearInput (): void { }	<i>Not supported</i>
clearOutput (): void { }	<i>Not supported</i>
directIO (command: <i>int32</i>, inout data: <i>int32</i>, inout obj: <i>object</i>): void { raises-exception, use after open }	1.0
resetStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
retrieveStatistics (inout statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
updateStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8

Specific

<i>Name</i>	<i>Version</i>
openDrawer (): void { raises-exception, use after open, enable }	1.0 <i>Note</i>
waitForDrawerClose (beepTimeout: <i>int32</i>, beepFrequency: <i>int32</i>, beepDuration: <i>int32</i>, beepDelay: <i>int32</i>): void { raises-exception, use after open, enable }	1.0 <i>Note</i>

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

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>
upos::events::DataEvent		<i>Not Supported</i>	
upos::events::DirectIOEvent			1.0
EventNumber:	<i>int32</i>	{ read-only }	
Data:	<i>int32</i>	{ read-write }	
Obj:	<i>object</i>	{ read-write }	
upos::events::ErrorEvent		<i>Not Supported</i>	
upos::events::OutputCompleteEvent		<i>Not Supported</i>	
upos::events::StatusUpdateEvent			1.0
Status:	<i>int32</i>	{ read-only }	

General Information

The Cash Drawer programmatic name is “CashDrawer”.

Capabilities

The Cash Drawer Control has the following capability:

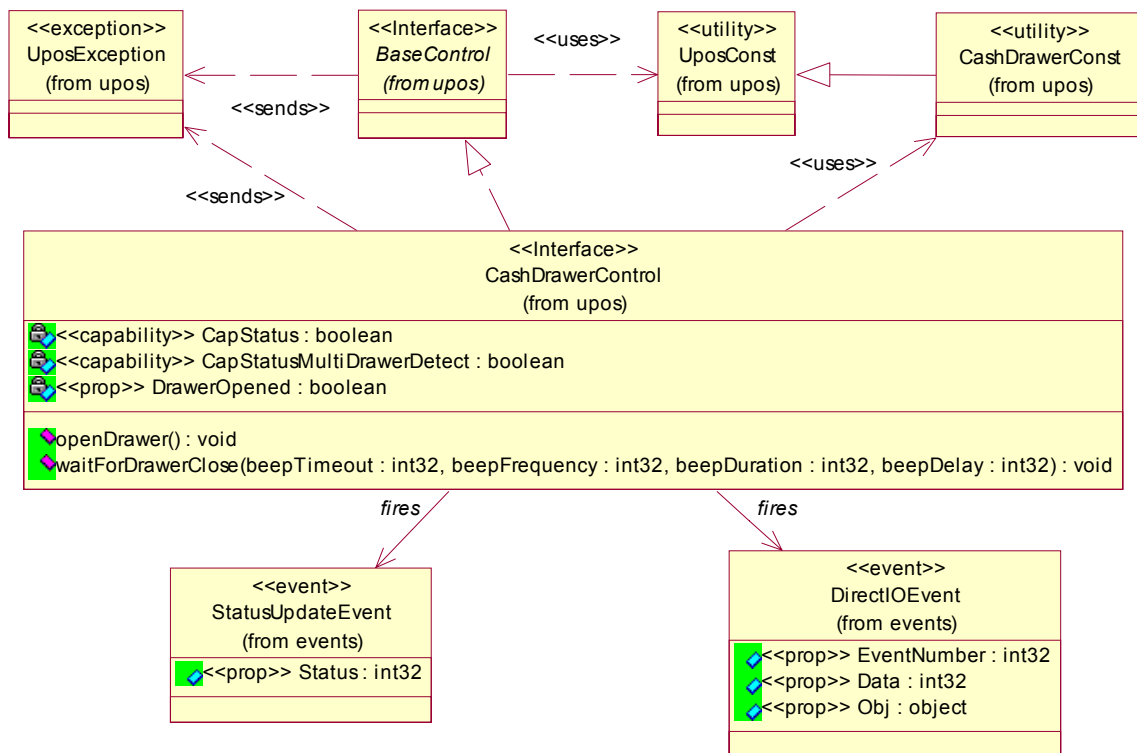
- Supports a command to “open” the cash drawer.

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.

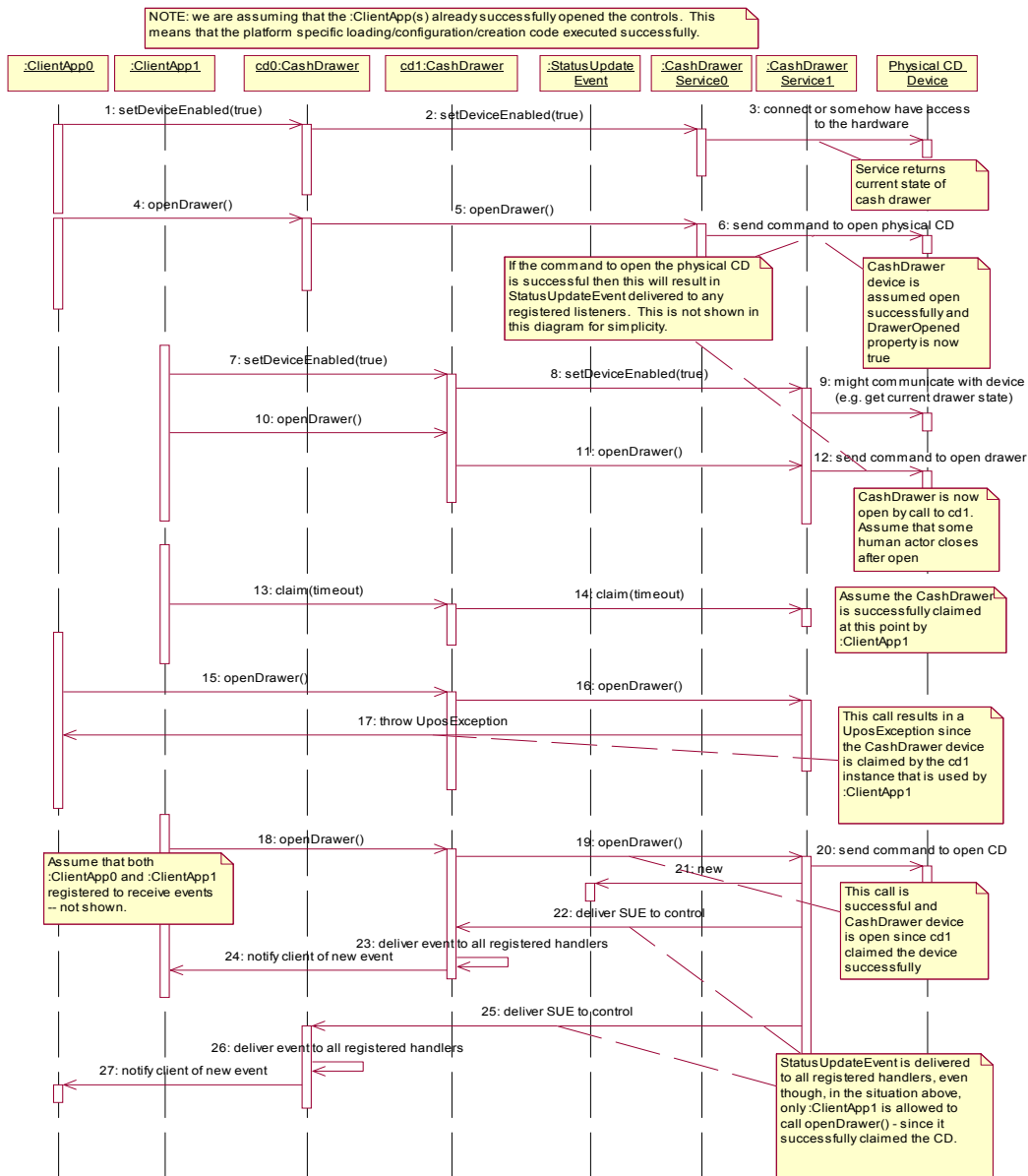
Cash Drawer Class Diagram *Updated in Release 1.8*

The following diagram shows the relationships between the Cash Drawer classes.



Cash Drawer Sequence Diagram *Updated in Release 1.8*

The following sequence diagram show the typical usage of a Cash Drawer open() → setDeviceEnabled(true) → getDrawerOpened() → openDrawer(); as well as showing the unique sharing model of the Cash Drawer device when used with multiple control instances open on the same physical device but by different applications.



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	If true, the drawer can report status. If false, the drawer is not able to determine whether the cash drawer is open or closed. 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 16.

CapStatusMultiDrawerDetect Property

Added in Release 1.5

Syntax	CapStatusMultiDrawerDetect: <i>boolean</i> { read-only , access after open }
Remarks	If true, the status unique to each drawer in a multiple cash drawer configuration ¹ can be reported. If false, the following possibilities exist: DrawerOpened: value of false indicates that there are no drawers open. 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. <i>Note:</i> A multiple cash drawer configuration is defined as one where a terminal or printer supports opening more than one cash drawer independently via the same channel or hardware port. A typical example is a configuration where a “Y” cable, connected to a single hardware printer port, has separate drawer open signal lines but the drawer open status from each of the drawers is “wired-or” together. It is not possible to determine which drawer is open. This property is only meaningful if CapStatus is true. 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 16.
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 16.
See Also	CapStatus Property, CapStatusMultiDrawerDetect Property.

Methods (UML operations)

openDrawer Method

Syntax	openDrawer (): void { raises-exception, use after open-enable }
Remarks	Opens the drawer.
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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-enable }										
	<table border="1"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><i>beepTimeout</i></td> <td>Number of milliseconds to wait before starting an alert beeper.</td> </tr> <tr> <td><i>beepFrequency</i></td> <td>Audio frequency of the alert beeper in hertz.</td> </tr> <tr> <td><i>beepDuration</i></td> <td>Number of milliseconds that the beep tone will be sounded.</td> </tr> <tr> <td><i>beepDelay</i></td> <td>Number of milliseconds between the sounding of beeper tones.</td> </tr> </tbody> </table>	Parameter	Description	<i>beepTimeout</i>	Number of milliseconds to wait before starting an alert beeper.	<i>beepFrequency</i>	Audio frequency of the alert beeper in hertz.	<i>beepDuration</i>	Number of milliseconds that the beep tone will be sounded.	<i>beepDelay</i>	Number of milliseconds between the sounding of beeper tones.
Parameter	Description										
<i>beepTimeout</i>	Number of milliseconds to wait before starting an alert beeper.										
<i>beepFrequency</i>	Audio frequency of the alert beeper in hertz.										
<i>beepDuration</i>	Number of milliseconds that the beep tone will be sounded.										
<i>beepDelay</i>	Number of milliseconds between the sounding of beeper tones.										
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 16.										
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 15, **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 63.

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 15, **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</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
CapStatisticsReporting:	<i>boolean</i>	{ read-only }	1.8	open
CapUpdateStatistics:	<i>boolean</i>	{ read-only }	1.8	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

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**

<i>Name</i>	<i>Version</i>
open (logicalDeviceName: <i>string</i>): void { raises-exception }	1.4
close (): void { raises-exception, use after open }	1.4
claim (timeout: <i>int32</i>): void { raises-exception, use after open }	1.4
release (): void { raises-exception, use after open, claim }	1.4
checkHealth (level: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.4
clearInput (): void { }	<i>Not supported</i>
clearOutput (): void { raises-exception, use after open, claim }	1.4
directIO (command: <i>int32</i>, inout data: <i>int32</i>, inout obj: <i>object</i>): void { raises-exception, use after open }	1.4
resetStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
retrieveStatistics (inout statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
updateStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8

Specific

<i>Name</i>	
accessDailyLog (sequenceNumber: <i>int32</i>, type: <i>int32</i>, timeout: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.4
authorizeCompletion (sequenceNumber: <i>int32</i>, amount: <i>currency</i>, taxOthers: <i>currency</i>, timeout: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.4
authorizePreSales (sequenceNumber: <i>int32</i>, amount: <i>currency</i>, taxOthers: <i>currency</i>, timeout: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.4
authorizeRefund (sequenceNumber: <i>int32</i>, amount: <i>currency</i>, taxOthers: <i>currency</i>, timeout: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.4
authorizeSales (sequenceNumber: <i>int32</i>, amount: <i>currency</i>, taxOthers: <i>currency</i>, timeout: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.4
authorizeVoid (sequenceNumber: <i>int32</i>, amount: <i>currency</i>, taxOthers: <i>currency</i>, timeout: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.4

authorizeVoidPreSales (sequenceNumber: int32, amount: currency, taxOthers: currency, timeout: int32): void { raises-exception, use after open, claim, enable }	1.4
checkCard (sequenceNumber: int32, timeout: int32): void { raises-exception, use after open, claim, enable }	1.4

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>
upos::events::DataEvent		<i>Not supported</i>	
upos::events::DirectIOEvent			1.4
EventNumber:	<i>int32</i>	{ read-only }	
Data:	<i>int32</i>	{ read-write }	
Obj:	<i>object</i>	{ read-write }	
upos::events::ErrorEvent			1.4
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			1.4
OutputID:	<i>int32</i>	{ read-only }	
upos::events::StatusUpdateEvent			1.4
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.

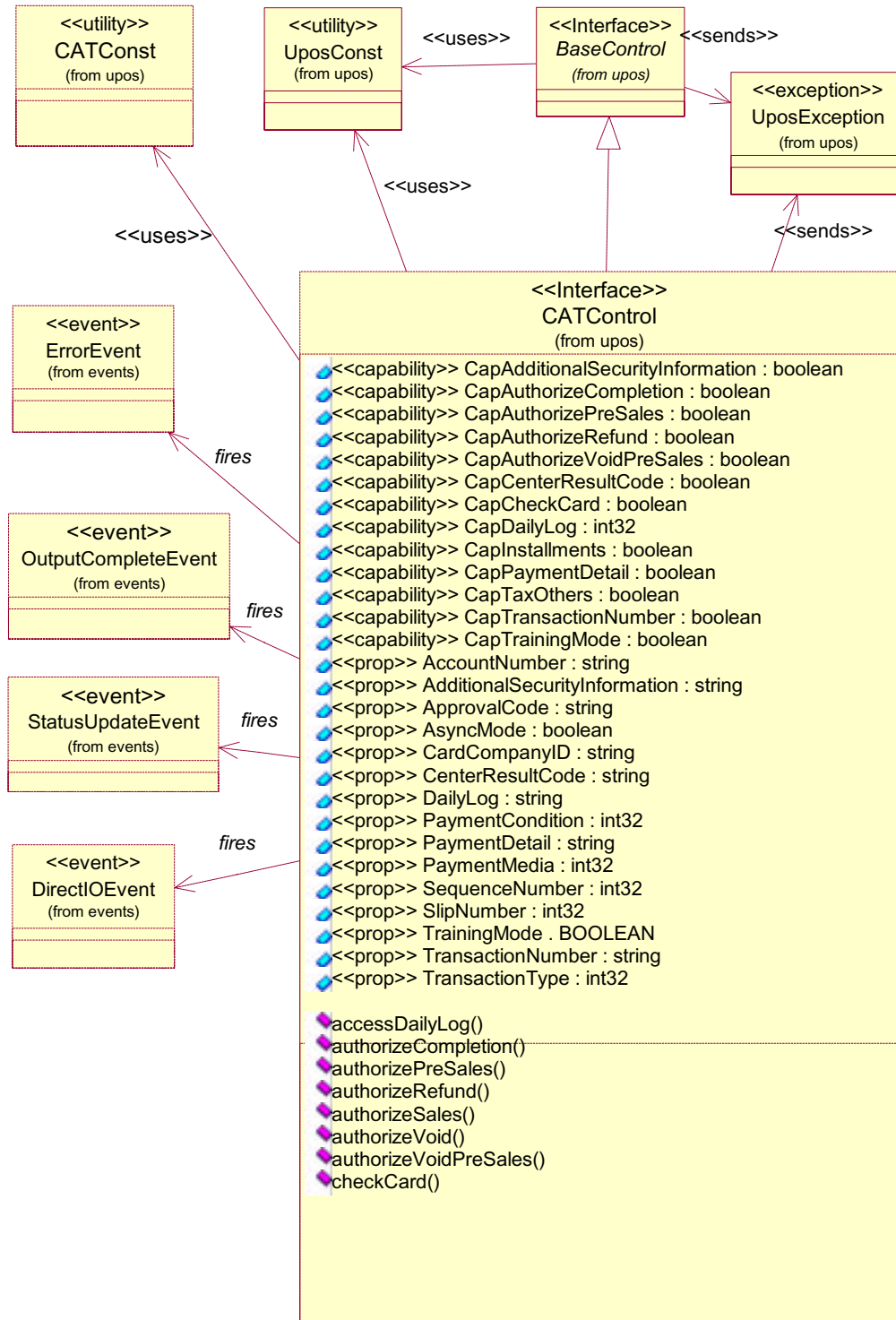
- **Card Check**
The transaction to perform a negative card file validation of the card presented by the customer. Typically negative card files contain card numbers that are known to fail approval. Therefore the Card Check operation removes then need for communication to the approval agency in some 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 one outstanding asynchronous request is allowed.
- The CAT control issues requests to the CAT device for different types of authorization by invoking the following methods.

Function	Method name	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 error 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 *UposException* 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 *UposException* 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**

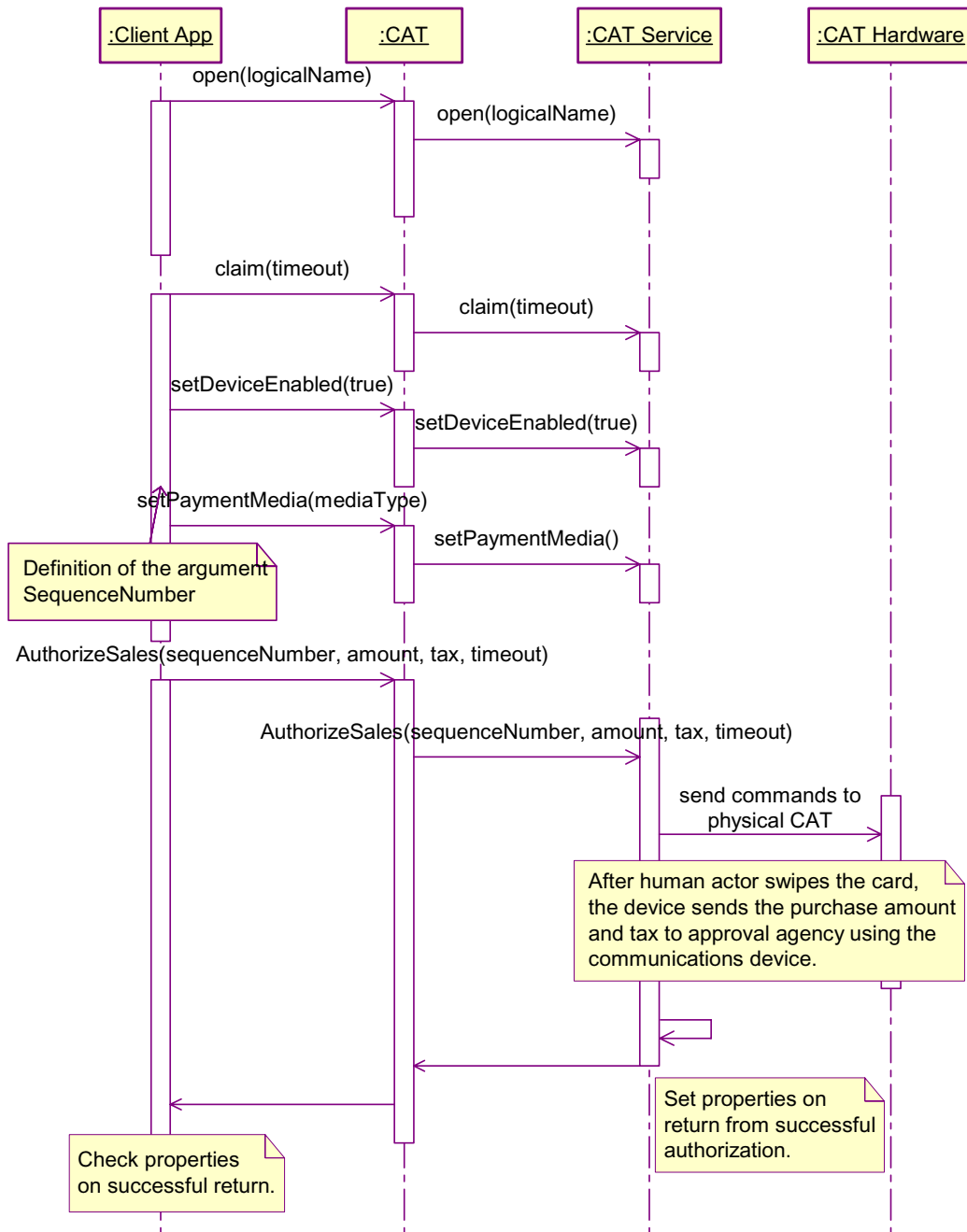
Device Sharing

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.

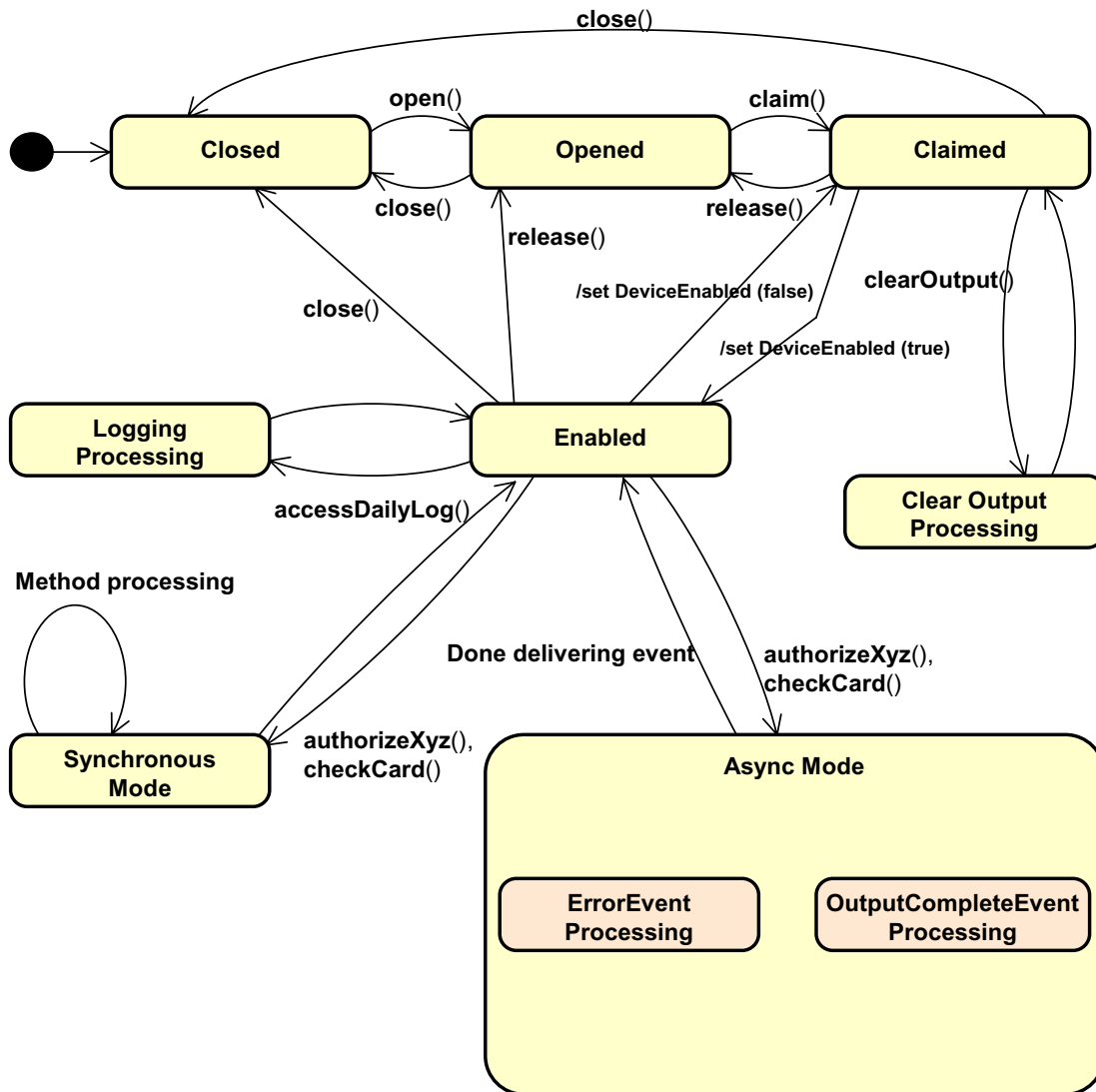
CAT Sequence Diagram *Added in Release 1.7*

This sequence diagram shows the typical synchronous usage of the **AuthorizeSales** process of the CAT device.



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 16.

AdditionalSecurityInformation Property **Updated in Release 1.7**

Syntax	AdditionalSecurityInformation: <i>string</i> { read-write, access after open } ¹
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 16.
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 16.

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 16.
See Also	Authorization Methods.

¹. In the OPOS environment, the format of this data depends upon the value of the **BinaryConversion** property. See **BinaryConversion** property on page A-28.

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 16.
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 16.
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 16.
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 16.
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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
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 16.

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 16.

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 16.

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 16.
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 16.
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 16.
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 16.

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 16.

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 `UposException` may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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
			UnifiedPOS 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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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 border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <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> </tbody> </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 16.								

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 16.

SlipNumber Property**Updated in Release 1.7**

Syntax	SlipNumber: <i>string</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 16.

TrainingMode Property

Syntax **TrainingMode:** *boolean* { read-write, access after open }

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

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

This property will be initialized to false by the **open** method.

Errors A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	CapTrainingMode is false.

TransactionNumber Property

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

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

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

Errors A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

TransactionType Property

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

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

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

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

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

Errors A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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 16.

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.

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 16.

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**: *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 Pre-authorization cancellation approval is intended.

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

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

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

Errors A UposException may be thrown when this method is invoked. For further information, see "Errors" on page 16.

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.

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 15, **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 CAT error has been detected and suitable response by the application is necessary to process the error condition.

Attributes This event contains the following attributes:

Attributes	Type	Description
<i>ErrorCode</i>	<i>int32</i>	The code which caused the error event. See <i>ErrorCode</i> for the values.
<i>ErrorCodeExtended</i>	<i>int32</i>	The extended code which caused the error event. See <i>ErrorCodeExtended</i> below for values.
<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 <i>ErrorResponse</i> below for values.

If *ErrorCode* is E_EXTENDED, *ErrorCodeExtended* 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 may set 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 21, Device Information Reporting Model on page 26.

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 21.

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. <i>Note that Release 1.3</i> added Power State Reporting with additional <i>Power reporting StatusUpdateEvent values</i> . See "StatusUpdateEvent" description on page 63.

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

See Also "Events" on page 15.

Check Scanner

This Chapter defines the Check Scanner device category.

Summary

Properties (UML attributes)

<i>Common</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
AutoDisable:	<i>boolean</i>	{ read-write }	1.7	open
CapPowerReporting:	<i>int32</i>	{ read-only }	1.7	open
CapStatisticsReporting:	<i>boolean</i>	{ read-only }	1.8	open
CapUpdateStatistics:	<i>boolean</i>	{ read-only }	1.8	open
CheckHealthText:	<i>string</i>	{ read-only }	1.7	open
Claimed:	<i>boolean</i>	{ read-only }	1.7	open
DataCount:	<i>int32</i>	{ read-only }	1.7	open
DataEventEnabled:	<i>boolean</i>	{ read-write }	1.7	open
DeviceEnabled:	<i>boolean</i>	{ read-write }	1.7	open & claim
FreezeEvents:	<i>boolean</i>	{ read-write }	1.7	open
OutputID:	<i>int32</i>	{ read-only }	1.7	Not Supported
PowerNotify:	<i>int32</i>	{ read-write }	1.7	open
PowerState:	<i>int32</i>	{ read-only }	1.7	open
State:	<i>int32</i>	{ read-only }	1.7	--
DeviceControlDescription:	<i>string</i>	{ read-only }	1.7	--
DeviceControlVersion:	<i>int32</i>	{ read-only }	1.7	--
DeviceServiceDescription:	<i>string</i>	{ read-only }	1.7	open
DeviceServiceVersion:	<i>int32</i>	{ read-only }	1.7	open
PhysicalDeviceDescription:	<i>string</i>	{ read-only }	1.7	open
PhysicalDeviceName:	<i>string</i>	{ read-only }	1.7	open

Properties (Continued)

<i>Specific</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
CapAutoGenerateFileID:	<i>boolean</i>	{ read-only }	1.7	open
CapAutoGenerateImageTagData:	<i>boolean</i>	{ read-only }	1.7	open
CapAutoSize:	<i>boolean</i>	{ read-only }	1.7	open
CapColor:	<i>int32</i>	{ read-only }	1.7	open
CapConcurrentMICR:	<i>boolean</i>	{ read-only }	1.7	open
CapDefineCropArea:	<i>boolean</i>	{ read-only }	1.7	open
CapImageFormat:	<i>int32</i>	{ read-only }	1.7	open
CapImageTagData:	<i>boolean</i>	{ read-only }	1.7	open
CapMICRDevice:	<i>boolean</i>	{ read-only }	1.7	open
CapStoreImageFiles:	<i>boolean</i>	{ read-only }	1.7	open
CapValidationDevice:	<i>boolean</i>	{ read-only }	1.7	open
Color:	<i>int32</i>	{ read-write }	1.7	open
ConcurrentMICR:	<i>boolean</i>	{ read-write }	1.7	open
CropAreaCount:	<i>int32</i>	{ read-only }	1.7	open
DocumentHeight:	<i>int32</i>	{ read-write }	1.7	open
DocumentWidth:	<i>int32</i>	{ read-write }	1.7	open
FileID:	<i>string</i>	{ read-write }	1.7	open
FileIndex:	<i>int32</i>	{ read-write }	1.7	open
ImageData:	<i>binary</i>	{ read-only }	1.7	open
ImageFormat:	<i>int32</i>	{ read-write }	1.7	open
ImageMemoryStatus:	<i>int32</i>	{ read-only }	1.7	open & claim
ImageTagData	<i>string</i>	{ read-write }	1.7	open
MapMode:	<i>int32</i>	{ read-write }	1.7	open
MaxCropAreas:	<i>int32</i>	{ read-only }	1.7	open
Quality:	<i>int32</i>	{ read-write }	1.7	open
QualityList:	<i>string</i>	{ read-only }	1.7	open
RemainingImagesEstimate:	<i>int32</i>	{ read-only }	1.7	open

Methods (UML operations)**Common**

<i>Name</i>	<i>Version</i>
open (logicalDeviceName: <i>string</i>): void { raises-exception }	1.7
close (): void { raises-exception, use after open }	1.7
claim (timeout: <i>int32</i>): void { raises-exception, use after open }	1.7
release (): void { raises-exception, use after open, claim }	1.7
checkHealth (level: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.7
clearInput (): void { raises-exception, use after open, claim }	1.7
clearOutput (): void { }	<i>Not supported</i>
directIO (command: <i>int32</i>, inout data: <i>int32</i>, inout obj: <i>object</i>): void { raises-exception, use after open, claim }	1.7
resetStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
retrieveStatistics (inout statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
updateStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8

Specific

beginInsertion (timeout: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.7
beginRemoval (timeout: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.7
clearImage (by: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.7
defineCropArea (cropAreaID: <i>int32</i>, x: <i>int32</i>, y: <i>int32</i>, cx: <i>int32</i>, cy: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.7
endInsertion (): void { raises-exception, use after open, claim, enable }	1.7
endRemoval (): void { raises-exception, use after open, claim, enable }	1.7
retrieveImage (cropAreaID: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.7
retrieveMemory(by: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.7
storeImage (cropAreaID: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.7

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>
upos::events::DataEvent			1.7
Status:	<i>int32</i>	{ read-only }	
upos::events::DirectIOEvent			1.7
EventNumber:	<i>int32</i>	{ read-only }	
Data:	<i>int32</i>	{ read-write }	
Obj:	<i>object</i>	{ read-write }	
upos::events::ErrorEvent			1.7
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		<i>Not Supported</i>	
upos::events::StatusUpdateEvent			1.7
Status:	<i>int32</i>	{ read-only }	

General Information

The Check Scanner programmatic name is “CheckScanner”.

Capabilities

The primary purpose of this device is to capture the image of a personal or business check for Electronic Check Conversion. However, other documents (vouchers, signature receipts, etc.) may be scanned if they fall within the capture size parameters of the Check Scanner. Therefore, in the description used in this standard the overall term “document” may be used to indicate the multiplicity of uses of which the device may be capable. When the term “check” is used, it should be viewed as a special form of a “document” as an example.

The Check Scanner Control has the following minimal set of capabilities:

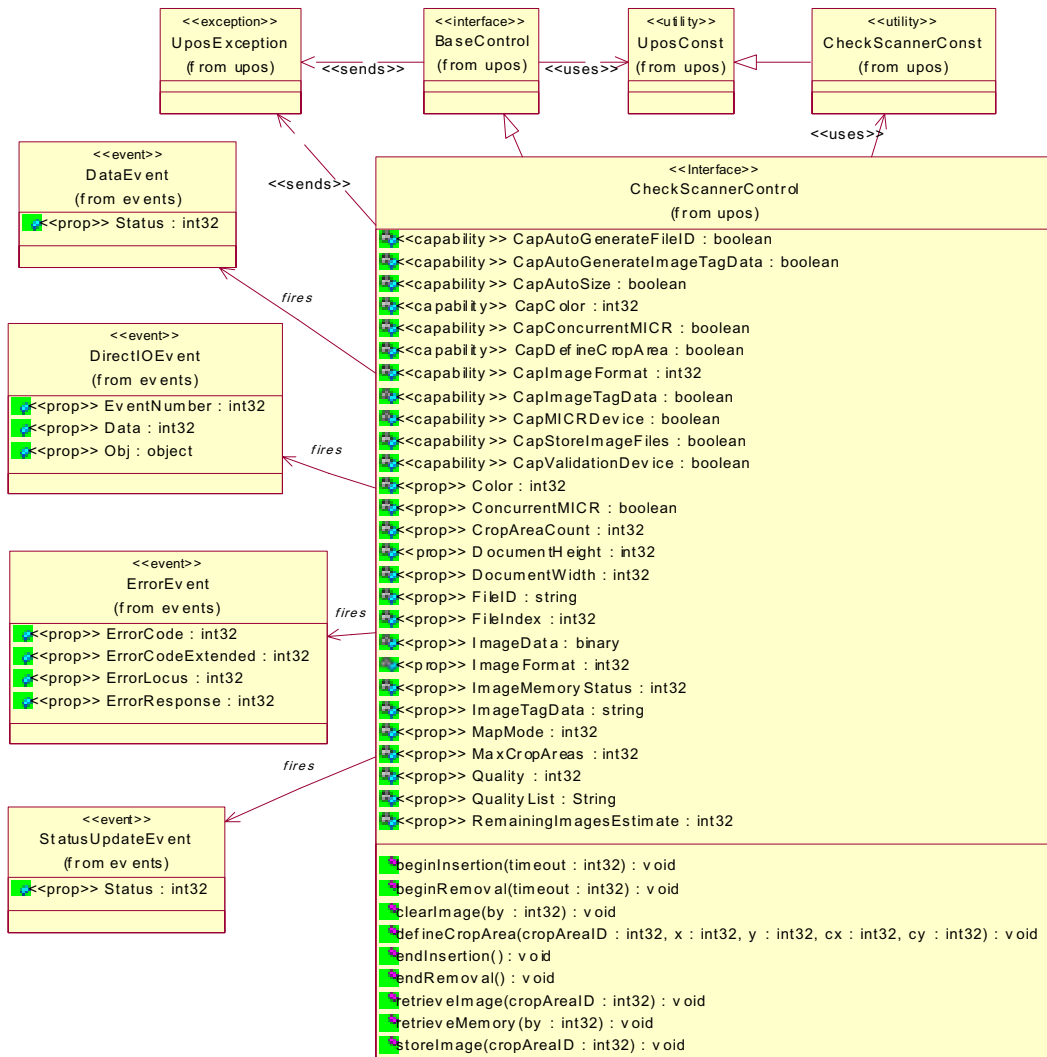
- Reads image data from a Check Scanner device.
- Has programmatic control of check insertion, reading, and removal. For some Check Scanner devices, this will require no processing in the Control since the device may automate many of these functions.

The Check Scanner Control may have the following additional capabilities:

- The Check Scanner may store successive check images in its hardware memory.
- Cropping of areas of interest within the check image may be supported by the Check Scanner to aid in the reduction of the memory needed to transmit or store the check image data.
- The **retrieveImage** data is deposited in the **ImageData** property in binary form.
- The Check Scanner may allow for retrieval of images stored in its hardware memory.
- The Check Scanner may support Image tag data information to identify the check image.
- The application reads the contents of **ImageData** property when it wants to further process the check image.
- The Check Scanner device may be physically attached to or incorporated into a check validation print device and/or a MICR device. If this is the case, once a check is inserted via Check Scanner Control methods, the check can still be used by the Printer and MICR Control prior to check removal.

Check Scanner Class Diagram

The following diagram shows the relationships between the Check Scanner classes.



Model

The Check Scanner Control follows the general “Input Model” . One point of difference is that the Check Scanner Control requires the execution of methods to insert and remove the check for processing. Therefore, this Control requires more than simply setting the **DataEventEnabled** property to true in order to receive data. The basic model is as follows:

- The Check Scanner Control is opened, claimed, and enabled.
- When the **beginInsertion** method is called, the Check Scanner is ready to read the check within the specified time as indicated by the time-out value. If the check is not inserted before the time-out value expires, a **UposException** is raised.
- In the event of a time-out, the Check Scanner device will remain in a state that allows a check to be inserted. The application may provide an operator prompt which requests that a check be inserted. Following this prompt, the application would then reissue the **beginInsertion** method and wait for the check to be inserted.
- Once a check is inserted, the **beginInsertion** method returns and the application calls the **endInsertion** method, which results in the Check Scanner device exiting the check insertion mode and causes the check image to be captured.
 - Following the **endInsertion** method, the scan image data is stored in a working buffer memory area and a **StatusUpdateEvent** will occur to indicate that a successful scan image process has taken place. No **DataEvent** is enqueued since data has not been transferred to the **ImageData** property at this point.
 - The application must use the **retrieveImage** method to retrieve the current scan image data. However, if the check image was not successfully captured by the device, the Control enqueues a **ErrorEvent** to indicate the capture was not successful.
 - If the **AutoDisable** property is true, then the device is automatically disabled when the image is successfully captured.
 - An enqueued **DataEvent** can be delivered to the application when the **DataEventEnabled** property is true and other event delivery requirements are met. Just before delivering this event, the Control copies data into specific properties, and disables further data events by setting the **DataEventEnabled** property to false. This causes subsequent input data to be enqueued by the Control while the application processes the current input and associated properties. When the application has finished the current input and is ready for more data, it reenables events by setting **DataEventEnabled** to true.
 - If the **CapAutoSize** property is true, when the **DataEvent** is delivered, the height and width of the of entire captured image are automatically stored in the corresponding **DocumentHeight** and **DocumentWidth** properties. If the **CapAutoSize** property is false, the application must manually set the **DocumentHeight** and the **DocumentWidth** property values prior to the **beginInsertion** method being invoked.
 - If the application needs to retrieve the entire or a cropped portion of the captured image, the **retrieveImage** method is called. The image data is

sent from the device to the service and stored in the **ImageData** property. When the corresponding **DataEvent** is delivered, the current image or cropped image may be accessed by the application reading the image file contained in the **ImageData** property.

- If the **CapStoreImageFiles** property is true, then the current image, or cropped image, can be stored in the memory by using the **storeImage** method.
- Any previously stored image may be retrieved by using the **retrieveMemory** method. The stored image may be identified using the “*by*” parameter and requesting that the image be located by **FileID**, **FileIndex**, or **ImageTagData**.
- If **CapDefineCropArea** is true, then the application can use the **defineCropArea** method to define crop areas in the captured image.
- An **ErrorEvent** (or events) is (are) enqueued if the Control encounters an error while reading the check, and is delivered to the application when the **DataEventEnabled** property is true and other event delivery requirements are met.
- All input data enqueued by the Control may be deleted by calling the **clearInput** method.
- After processing the **endInsertion DataEvent**, the application may query the **CapMICRDevice** property to determine if the device supports Magnetic Ink Character Recognition. If **CapMICRDevice** property is true, then a MICR read function may be performed in a “single pass” or “multiple pass” cycle but prior to the check being removed from the device. If **CapConcurrentMICR** property is true, then the device is capable of supporting a “single pass” MICR read during an image scan. If **CapConcurrentMICR** property is true and **ConcurrentMICR** property is true, then the MICR data would be read and calling the MICR's **beginInsertion** and **endInsertion** methods would not be needed to reposition the check for MICR reading.
- Additionally, after processing a **DataEvent**, the application should query the **CapValidationDevice** property to determine if validation printing can be performed on the check prior to check removal. If this property is true, the application may call the Printer Control's **beginInsertion** and **endInsertion** methods. This positions the check for validation printing. The Printer Control's validation printing methods can then be used to perform validation printing.
- If the **CapImageTagData** property is true, then an identifying name, for example the transaction number, date and time, or some other naming element, could be used to identify the image data. The identifying name is set in **ImageTagData** property.
- Once the check is no longer needed in the device, the application must call the **beginRemoval** method of the Check Scanner, the MICR (if **CapMICRDevice** is true), or the POS Printer (if **CapValidationDevice** is true), also specifying a timeout value. This method will raise a **UposException** if the check is not removed within the timeout period. In this case, the application may perform any additional prompting prior to calling the method again. Once the check is removed, the application should call the same device's **endRemoval** method to take the device out of removal mode.
- In order to accommodate many different Check Scanning devices, the

application should follow the above sequence of method calls even though the device may not physically require one or more of the methods. An example may be a Check Scanner that is “auto armed” and is capable of detecting a check present and initiating a Check Scan and MICR read cycle automatically. In this case the **beginInsertion**, **endInsertion**, **beginRemoval**, and **endRemoval** method calls may actually do no more than return from the Service.

- The model assumes that the device has a work area that can be used in the following ways:
 - When a document is scanned its image will be loaded as raw data into this work area. When the **retrieveImage** method is invoked the data from the work area may be modified by a previously defined crop area, as specified by the *cropAreaID* parameter, and loaded into the **ImageData** property. The work area will still contain the original scanned image data. Additional **retrieveImage** method calls using different crop area criteria can then be accomplished to load the **ImageData** property.
 - The work area contains image data either from a recently scanned image or as a result of a **retrieveMemory** method. Prior to invoking the **storeImage** method, the **FileIndex** property is set to the correct index number (as maintained by the service) and if used, the **FileID** and/or **ImageTagData** properties are set. When the **storeImage** method is invoked the data from the work area may be modified by a previously defined crop area, as specified by the *cropAreaID* parameter, and stored in the device memory. The work area will still contain the original scanned image data. Additional **storeImage** method calls using different crop area criteria can then be accomplished to store the image data in the device’s memory. The **RemainingImagesEstimate** property is adjusted to reflect the approximate number additional images that may be stored in the device memory based upon the file size history of previously stored images.
 - When the **retrieveMemory** method is invoked, the work area is loaded with an image data file that was previously stored in the device memory. Either the **FileIndex**, **FileID**, or **ImageTagData** may be used to locate the previously stored image. The **ImageData** property is also loaded with the retrieved image data.
- In order to accommodate the various storage and retrieval architectures that are in use for the Check Scanner device class, the model has been designed to allow for three different addressing ways to locate previously stored image data: **FileIndex**, **FileID**, and **ImageTagData**.
 - The **FileIndex** is an addressing scheme that is automatically provided by the service to physically store and retrieve the file data. The definition of file data in this case includes any and/or all of the following: image data, tag data information (that is appended and included with the image data file), and a file identification (a file name associated with the image data file). The **FileIndex** is only used by the service to save and retrieve the scan data and its associated data elements.

- The **FileID** is a “file name” that may be provided automatically by the hardware device or the service. It also may be populated by the application prior to a **storeImage** method being called. Once created it remains with the **ImageData** and can be used to randomly locate a specific file for uploading to the POS system and post processing applications.
- The **ImageTagData** is a set of information about the image that has been scanned. Typically it contains information about when the image was captured (Date and Time, for instance), Store number, Lane Number, Clerk identification, etc. This data is typically pre or post appended to the **ImageData** and remains a part of the combined data file as a record of the origin of the data.

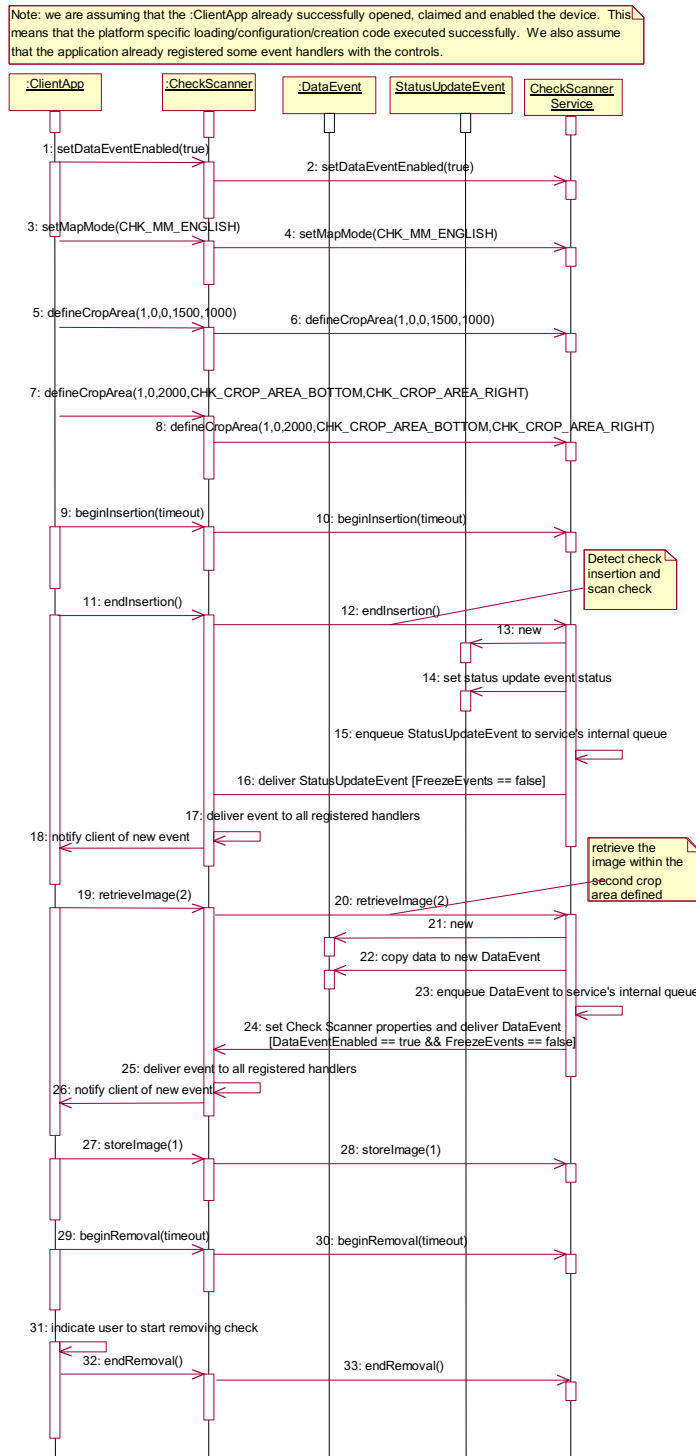
Device Sharing

The Check Scanner is an exclusive-use device, and adheres to the following constraints:

- 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.

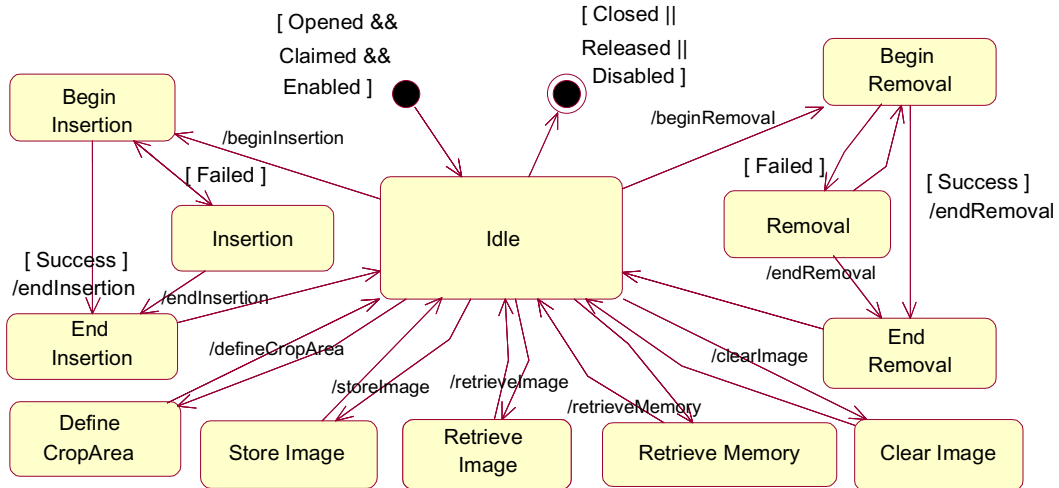
Check Scanner Sequence Diagram

The following sequence diagram shows the typical usage of the Check Scanner device.



Check Scanner State Diagram

The following diagram depicts the Check Scanner control device model.



Properties (UML attributes)

CapAutoGenerateFileID Property

Syntax	CapAutoGenerateFileID: <i>boolean</i> { read-only, access after open }
Remarks	<p>This capability indicates the ability of the device to automatically generate a file name that can be used to reference the file containing the captured image.</p> <p>If CapAutoGenerateFileID is true, then the device can automatically create a file name for the captured image file.</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 16.
See Also	FileID Property.

CapAutoGenerateImageTagData Property

Syntax	CapAutoGenerateImageTagData: <i>boolean</i> { read-only, access after open }
Remarks	<p>This capability indicates the ability of the device to automatically generate tag data used in reference to the image file for the captured image.</p> <p>If CapAutoGenerateImageTagData is true, then the device can automatically create image tag data which can be appended to the image file to provide information about the captured image.</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 16.
See Also	ImageTagData Property.

CapAutoSize Property

Syntax	CapAutoSize: <i>boolean</i> { read-only, access after open }
Remarks	<p>This capability indicates the ability of the device to determine the height and width of the document automatically.</p> <p>If CapAutoSize is true, then the height and width of the scanned document will be automatically placed in the DocumentHeight and DocumentWidth properties when the image is captured.</p> <p>If CapAutoSize is false, the height and width of the document can be manually set in the DocumentHeight and DocumentWidth properties by the application prior to scanning an image.</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 16.
See Also	DocumentHeight Property, DocumentWidth Property.

CapColor Property

Syntax	CapColor: <i>int32</i> { read-only, access after open }												
Remarks	<p>This capability indicates if this device supports image formats other than bi-tonal.</p> <p>CapColor is a logical OR combination of any of the following values:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>CHK_CCL_MONO</td> <td>Bi-tonal (B/W)</td> </tr> <tr> <td>CHK_CCL_GRAYSCALE</td> <td>Gray scale</td> </tr> <tr> <td>CHK_CCL_16</td> <td>16 Colors</td> </tr> <tr> <td>CHK_CCL_256</td> <td>256 Colors</td> </tr> <tr> <td>CHK_CCL_FULL</td> <td>Full colors</td> </tr> </tbody> </table> <p>This property is initialized by the open method.</p>	Value	Meaning	CHK_CCL_MONO	Bi-tonal (B/W)	CHK_CCL_GRAYSCALE	Gray scale	CHK_CCL_16	16 Colors	CHK_CCL_256	256 Colors	CHK_CCL_FULL	Full colors
Value	Meaning												
CHK_CCL_MONO	Bi-tonal (B/W)												
CHK_CCL_GRAYSCALE	Gray scale												
CHK_CCL_16	16 Colors												
CHK_CCL_256	256 Colors												
CHK_CCL_FULL	Full colors												
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.												
See Also	Color Property.												

CapConcurrentMICR Property

Syntax	CapConcurrentMICR: <i>boolean</i> { read-only, access after open }
Remarks	<p>This capability indicates if this device supports a Magnetic Ink Character Recognition read during the image scanning process.</p> <p>If CapConcurrentMICR is true, a check's MICR data can be captured during a check scanning cycle (single pass scanning). For devices that are both a Check Scanner device and a MICR reader device, following a check scan the device will automatically pass the MICR data to the MICR Service. The check will not need to be re-read during the MICR beginInsertion and endInsertion methods.</p> <p>If CapConcurrentMICR is false, then it would be necessary to read the MICR data (if the device supports MICR reading) by using the MICR beginInsertion and endInsertion methods. Usually the MICR read is performed prior to the Check Scanning process.</p> <p>This property has no meaning if the CapMICRDevice property 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 16.
See Also	CapMICRDevice Property, ConcurrentMICR Property.

CapDefineCropArea Property

Syntax	CapDefineCropArea: <i>boolean</i> { read-only, access after open }
Remarks	<p>This capability indicates if this device supports a feature that allows cropping of areas of interest within the scan image area defined by the DocumentHeight and DocumentWidth properties.</p> <p>If CapDefineCropArea is true, one or more cropping areas are allowed; otherwise it is set to be 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 16.
See Also	CropAreaCount Property, MaxCropAreas Property, defineCropArea Method.

CapImageFormat Property

Syntax	CapImageFormat: <i>int32</i> { read-only, access after open }												
Remarks	<p>This capability indicates the image file formats that this device supports. The image data is stored in the ImageData property using one of the following formats supported by the CapImageFormat Property:</p> <p>CapImageFormat is a logical OR combination of any of the following values:</p> <table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>CHK_CIF_NATIVE</td> <td>Hardware native format</td> </tr> <tr> <td>CHK_CIF_TIFF</td> <td>TIFF format</td> </tr> <tr> <td>CHK_CIF_BMP</td> <td>BMP format</td> </tr> <tr> <td>CHK_CIF_JPEG</td> <td>JPEG format</td> </tr> <tr> <td>CHK_CIF_GIF</td> <td>GIF format</td> </tr> </tbody> </table> <p>This property is initialized by the open method.</p>	Value	Meaning	CHK_CIF_NATIVE	Hardware native format	CHK_CIF_TIFF	TIFF format	CHK_CIF_BMP	BMP format	CHK_CIF_JPEG	JPEG format	CHK_CIF_GIF	GIF format
Value	Meaning												
CHK_CIF_NATIVE	Hardware native format												
CHK_CIF_TIFF	TIFF format												
CHK_CIF_BMP	BMP format												
CHK_CIF_JPEG	JPEG format												
CHK_CIF_GIF	GIF format												
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.												
See Also	ImageFormat Property.												

CapImageTagData Property

Syntax	CapImageTagData: <i>boolean</i> { read-only, access after open }
Remarks	<p>This capability indicates if this device has the ability to utilize tag names to identify its scanned images.</p> <p>If CapImageTagData is true, then the device can set tag data, as defined by the ImageTagData property, to the image data file stored in the ImageData property.</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 16.
See Also	ImageTagData Property, retrieveImage Method, storeImage Method.

CapMICRDevice Property

Syntax	CapMICRDevice: <i>boolean</i> { read-only, access after open }
Remarks	<p>This capability indicates if this device supports a check MICR read function.</p> <p>If CapMICRDevice is true, then the device supports a MICR read function in addition to check scanning.</p> <p>If CapConcurrentMICR is true, a check's MICR data can be captured during a check scanning cycle (single pass scanning). For devices that are both a Check Scanner device and a MICR reader device, following a check scan the device will automatically pass the MICR data to the MICR service. The check will not need to be re-read during the MICR beginInsertion and endInsertion methods.</p> <p>If CapConcurrentMICR property is false, then it would be necessary to read the MICR data by using the MICR beginInsertion and endInsertion methods. In this case the MICR read is usually performed prior to the Check Scanning process.</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 16.
See Also	CapConcurrentMICR Property, ConcurrentMICR Property.

CapStoreImageFiles Property

Syntax	CapStoreImageFiles: <i>boolean</i> { read-only, access after open }
Remarks	<p>This capability indicates if this device has the ability to store check images in its hardware memory.</p> <p>If CapStoreImageFiles is true, one or more images can be stored in the memory provided by the device by using the storeImage method.</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 16.
See Also	retrieveImage Method, storeImage Method.

CapValidationDevice Property

Syntax	CapValidationDevice: <i>boolean</i> { read-only, access after open }
Remarks	<p>This capability indicates if this device has the ability to perform a validation print function on the check using a print station.</p> <p>If CapValidationDevice is true, a check does not have to be removed from the Check Scanner device prior to performing validation printing. For devices that are both a Check Scanner device as well as a POS Printer, the device will automatically position the check for validation printing after successfully performing a Check Scanner read. Either the Check Scanner Control's or the POS Printer Control's beginRemoval and endRemoval methods may be called to remove the check once the process is complete.</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 16.

Color Property

Syntax	Color: <i>int32</i> { read-write, access after open }												
Remarks	<p>This property is used to select the image scan mode for subsequent document scan operations. The available options may be affected by the current file type as specified by the ImageFormat property. Certain file types may not work with all the "colors" that the device may support. It is up to the application to insure that the proper Color and ImageFormat properties are compatible. Changing the Color property will not affect any previously stored data currently residing in the ImageData property.</p> <p>It may contain one of the following values:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>CHK_CL_MONO</td> <td>Bi-tonal (B/W)</td> </tr> <tr> <td>CHK_CL_GRAYSCALE</td> <td>Gray scale</td> </tr> <tr> <td>CHK_CL_16</td> <td>16 Colors</td> </tr> <tr> <td>CHK_CL_256</td> <td>256 Colors</td> </tr> <tr> <td>CHK_CL_FULL</td> <td>Full color</td> </tr> </tbody> </table> <p>This property is initialized by the open method.</p>	Value	Meaning	CHK_CL_MONO	Bi-tonal (B/W)	CHK_CL_GRAYSCALE	Gray scale	CHK_CL_16	16 Colors	CHK_CL_256	256 Colors	CHK_CL_FULL	Full color
Value	Meaning												
CHK_CL_MONO	Bi-tonal (B/W)												
CHK_CL_GRAYSCALE	Gray scale												
CHK_CL_16	16 Colors												
CHK_CL_256	256 Colors												
CHK_CL_FULL	Full color												
Errors	A UposException may be thrown when this property is accessed. For further information, see "Errors" on page 16.												
See Also	CapColor Property, ImageFormat Property.												

ConcurrentMICR Property

Syntax	ConcurrentMICR: <i>boolean</i> { read-write, access after open }
Remarks	<p>This property indicates whether a MICR read should be performed at the same time the check image is captured (single pass operation).</p> <p>This property has no meaning if the CapMICRDevice is false.</p> <p>If ConcurrentMICR is true, a check's MICR data is captured during a check scanning cycle (single pass scanning). For devices that are both a Check Scanner device and a MICR reader device, following a check scan the device will automatically pass the MICR data to the MICR Service. The check will not need to be re-read during the MICR beginInsertion and endInsertion methods.</p> <p>If ConcurrentMICR is false and MICR data is required, then it is necessary to read MICR data by using the MICR beginInsertion and endInsertion method calls. In this case the MICR read is usually performed prior to the Check Scanning process.</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 16.
See Also	CapConcurrentMICR Property, CapMICRDevice Property.

CropAreaCount Property

Syntax	CropAreaCount: <i>int32</i> { read-only, access after open }
Remarks	<p>This property indicates the number of Crop areas that have been defined which may be applied to the captured image.</p> <p>If CapDefineCropArea is false, then this property is always zero.</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 16.
See Also	CapDefineCropArea Property, MaxCropAreas Property, defineCropArea Method.

DocumentHeight Property

Syntax	DocumentHeight: <i>int32</i> { read-write, access after open }
Remarks	<p>This property is used to define the height of the document scanned or the height of a document to scan. It is expressed in the unit of measure as defined by the MapMode property.</p> <p>If CapAutoSize is true, then the height of the scanned document will be automatically placed in the DocumentHeight property when the image is captured.</p> <p>If CapAutoSize is false, the height of the document can be manually set in the DocumentHeight property by the application prior to scanning a document.</p> <p>This property is initialized to the maximum height supported by the device 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 16.
See Also	CapAutoSize Property, MapMode Property.

DocumentWidth Property

Syntax	DocumentWidth: <i>int32</i> { read-write, access after open }
Remarks	<p>This property is used to define the width of the document scanned or the width of a document to scan. It is expressed in the unit of measure as defined by the MapMode property.</p> <p>If CapAutoSize is true, then the width of the scanned document will be automatically placed in the DocumentWidth property when the image is captured.</p> <p>If CapAutoSize is false, the width of the document can be manually set in the DocumentWidth property by the application prior to scanning an image.</p> <p>This property is initialized to the maximum width supported by the device 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 16.
See Also	CapAutoSize Property, MapMode Property.

FileID Property

Syntax	FileID: <i>string</i> { read-write, access after open }
Remarks	<p>This property is used to store a “file name” associated with the image data file. If the application chooses to create the data for this property, it must set the FileID property prior to calling the storeImage method.</p> <p>After a retrieveMemory method call the FileID property will be set to the image data file name if available, otherwise it will be set to a NULL (0x00). Its value is set prior to a DataEvent being delivered to the application.</p> <p>If the CapAutoGenerateFileID property is true then the FileID will automatically be generated by the hardware device or the service when the image is scanned.</p> <p>This property is initialized to NULL (0x00) by the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	CapAutoGenerateFileID Property, retrieveImage Method, retrieveMemory Method, storeImage Method.

FileIndex Property

Syntax	FileIndex: <i>int32</i> { read-write, access after open }
Remarks	<p>This property is used to store a file location reference to the image data file when either the storeImage or retrieveMemory methods are called. Its value is set prior to a DataEvent being delivered to the application.</p> <p>The FileIndex property is used only by the service in conjunction with the device to store and locate an image data file.</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 16.
See Also	clearImage Method, retrieveImage Method, retrieveMemory Method, storeImage Method.

ImageData Property

Syntax	ImageData: <i>binary</i> { read-only, access after open } ¹
Remarks	This property is used to store the image data after the retrieveImage or retrieveMemory methods are called. If no image data was available, the ImageData property will be set to NULL (0x00). Its value is set prior to a DataEvent being delivered to the application. This property is initialized to NULL (0x00) by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	retrieveImage Method, DataEvent .

ImageFormat Property

Syntax	ImageFormat: <i>int32</i> { read-write, access after open }												
Remarks	This property is used to define the data format of the image file that the device will use when it captures an image. The availability of acceptable file types is specified in the CapImageFormat property. The ImageFormat property must be set before a document is scanned. Any previously stored data in the ImageData property will not be affected by changing the value of the ImageFormat property. If the device provides support, it may be one of the following values: <table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>CHK_IF_NATIVE</td> <td>Hardware native format</td> </tr> <tr> <td>CHK_IF_TIFF</td> <td>TIFF format</td> </tr> <tr> <td>CHK_IF_BMP</td> <td>BMP format</td> </tr> <tr> <td>CHK_IF_JPEG</td> <td>JPEG format</td> </tr> <tr> <td>CHK_IF_GIF</td> <td>GIF format</td> </tr> </tbody> </table> The default value of this property is CHK_IF_TIFF. This property is initialized by the open method.	Value	Meaning	CHK_IF_NATIVE	Hardware native format	CHK_IF_TIFF	TIFF format	CHK_IF_BMP	BMP format	CHK_IF_JPEG	JPEG format	CHK_IF_GIF	GIF format
Value	Meaning												
CHK_IF_NATIVE	Hardware native format												
CHK_IF_TIFF	TIFF format												
CHK_IF_BMP	BMP format												
CHK_IF_JPEG	JPEG format												
CHK_IF_GIF	GIF format												
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.												
See Also	CapImageFormat Property, Color Property, DataEvent .												

¹. In the OPOS environment, the format of this data depends upon the value of the **BinaryConversion** property. See **BinaryConversion** property on page A-28.

ImageMemoryStatus Property

Syntax	ImageMemoryStatus: <i>int32</i> { read-only, access after open-claim }								
Remarks	<p>This property is used to indicate the current memory availability status if the device has the ability to store multiple image files. The ImageMemoryStatus value is only valid if the CapStoreImageFiles is true.</p> <p>The following values are supported.</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>CHK_IMS_EMPTY</td> <td>The image memory is empty.</td> </tr> <tr> <td>CHK_IMS_OK</td> <td>The image memory is has storage available.</td> </tr> <tr> <td>CHK_IMS_FULL</td> <td>The image memory is full.</td> </tr> </tbody> </table>	Value	Meaning	CHK_IMS_EMPTY	The image memory is empty.	CHK_IMS_OK	The image memory is has storage available.	CHK_IMS_FULL	The image memory is full.
Value	Meaning								
CHK_IMS_EMPTY	The image memory is empty.								
CHK_IMS_OK	The image memory is has storage available.								
CHK_IMS_FULL	The image memory is full.								
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.								
See Also	CapStoreImageFiles Property, storeImage Method.								

ImageTagData Property

Syntax	ImageTagData: <i>string</i> { read-write, access after open }
Remarks	<p>This property is used to define a string that specifies the tag name for the captured image data. It may be specified by the application or auto-generated by the Check Scanner device.</p> <p>If the application chooses to create the data for this property, it must set the ImageTagData property prior to calling the storeImage method. After a retrieveMemory method call, the ImageTagData property will be set if available, otherwise it will be set to a NULL (0x00). Its value is set prior to a DataEvent being delivered to the application.</p> <p>If the CapAutoGenerateImageTagData property is true, the ImageTagData will automatically be generated by the hardware device or the service when the image is scanned.</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 16.
See Also	CapAutoGenerateImageTagData Property, retrieveImage Method, retrieveMemory Method, storeImage Method.

MapMode Property

Syntax	MapMode: <i>int32</i> { read-write, access after open }										
Remarks	<p>This property is used to specify the units of measure that are currently valid for the Check Scanner.</p> <p>The mapping mode defines the unit of measure used by other properties, such as the DocumentHeight and DocumentWidth properties.</p> <p>The following units of measure may be selected for storing the image:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>CHK_MM_DOTS</td> <td>The scanner's dot width.</td> </tr> <tr> <td>CHK_MM_TWIPS</td> <td>1/1440 of an inch.</td> </tr> <tr> <td>CHK_MM_ENGLISH</td> <td>0.001 inch.</td> </tr> <tr> <td>CHK_MM_METRIC</td> <td>0.01 millimeter.</td> </tr> </tbody> </table> <p>The value of MapMode is initialized to CHK_MM_ENGLISH when the device is first enabled following the open method.</p>	Value	Meaning	CHK_MM_DOTS	The scanner's dot width.	CHK_MM_TWIPS	1/1440 of an inch.	CHK_MM_ENGLISH	0.001 inch.	CHK_MM_METRIC	0.01 millimeter.
Value	Meaning										
CHK_MM_DOTS	The scanner's dot width.										
CHK_MM_TWIPS	1/1440 of an inch.										
CHK_MM_ENGLISH	0.001 inch.										
CHK_MM_METRIC	0.01 millimeter.										
Errors	A UposException may be thrown when this property is accessed. For further information, see "Errors" on page 16.										
See Also	DocumentHeight Property, DocumentWidth Property, defineCropArea Method.										

MaxCropAreas Property

Syntax	MaxCropAreas: <i>int32</i> { read-only, access after open }
Remarks	<p>This property is used to specify the maximum number of crop areas that the device can support.</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 16.
See Also	CapDefineCropArea Property, CropAreaCount Property, defineCropArea Method.

Quality Property

Syntax	Quality: <i>int32</i> { read-write, access after open }
Remarks	<p>This property is used to set the resolution of the device when a scan image is to take place. It is defined as a dpi (dots per inch) value.</p> <p>Any previously stored data in ImageData property will not be affected when the Quality property value is changed.</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 16.
See Also	QualityList Property.

QualityList Property

Syntax	QualityList: string { read-only, access after open }
Remarks	<p>This property is used to define the resolutions that the Check Scanner is capable of supporting.</p> <p>The string data consists of comma separated values that indicate the available scanning resolutions that the device supports measured in dots per inch (dpi). An empty string indicates that resolution is not selectable.</p> <p>An example might be “160,320”, which indicates that the device supports 160 dpi and 320 dpi.</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 16.
See Also	Quality Property.

RemainingImagesEstimate Property

Syntax	RemainingImagesEstimate: <i>int32</i> { read-only, access after open }
Remarks	<p>This property is used to provide a “best guess” estimate of the remaining number of images that can be stored. It is updated after every new image is stored or cleared from the device’s available memory. The RemainingImagesEstimate along with the ImageMemoryStatus properties are intended to be used by the application to monitor the amount of available image storage.</p> <p>This property is initialized to a “best guess” estimate of the total number of image files that can be stored in the device’s memory by the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	ImageMemoryStatus Property.

Methods (UML operations)

beginInsertion Method

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

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

If zero, the method tries to begin insertion mode, then returns immediately if successful. otherwise a `UposException` is raised. If `UPOS_FOREVER (-1)`, the method tries to begin insertion mode, then waits as long as needed until either the check is inserted or an error occurs.

Remarks Called to initiate the document insertion process.

When called, the Check Scanner is made ready to receive a check by opening the Check Scanner's check handling "jaws" or activating a Check Scanner's check insertion mode. This method is paired with the **endInsertion** method for controlling the check insertion. Although some Check Scanner devices do not require this sort of processing, the application should still use these methods to ensure application portability across different Check Scanner devices.

If the Check Scanner device cannot be placed into insertion mode, a `UposException` is raised. Otherwise, check insertion is monitored until either:

- The check is successfully inserted.
- The check is not inserted before *timeout* milliseconds have elapsed, or an error is reported by the Check Scanner device. In this case, a `UposException` is raised, The Check Scanner device remains in check insertion mode. This allows an application to perform some user interaction and reissue the **beginInsertion** method without altering the Check Scanner check handling mechanism.

Errors A `UposException` may be thrown when this method is invoked. For further information, see "Errors" on page 16.

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

Value	Meaning
E_BUSY	If the Check Scanner is a combination device, the peer device may be busy.
E_ILLEGAL	An invalid <i>timeout</i> parameter was specified.
E_TIMEOUT	The specified time has elapsed without the check being properly inserted.

See Also **beginRemoval** Method, **endInsertion** Method, **endRemoval** Method.

beginRemoval Method

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

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

If zero, the method tries to begin removal mode, then returns immediately if successful. otherwise a `UposException` is raised. If `UPOS_FOREVER (-1)`, the method tries to begin removal mode, then waits as long as needed until either the check is removed or an error occurs.

Remarks Called to initiate the check removal processing.

When called, the Check Scanner is made ready to remove a check by opening the Check Scanner's check handling "jaws" or activating a Check Scanner's check ejection mode. This method is paired with the **endRemoval** method for controlling check removal. Although some Check Scanner devices do not require this sort of processing, the application should still use these methods to ensure application portability across different Check Scanner devices.

If the Check Scanner device cannot be placed into removal or ejection mode, a `UposException` is raised. Otherwise, check removal is monitored until either:

- The check is successfully removed.
- The check is not removed before *timeout* milliseconds have elapsed, or an error is reported by the Check Scanner device. In this case, a `UposException` is raised, The Check Scanner device remains in check removal mode. This allows an application to perform some user interaction and reissue the **beginRemoval** method without altering the Check Scanner check handling mechanism.

Errors A `UposException` may be thrown when this method is invoked. For further information, see "Errors" on page 16.

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

Value	Meaning
E_BUSY	If the Check Scanner is a combination device, the peer device may be busy.
E_ILLEGAL	An invalid <i>timeout</i> parameter was specified.
E_TIMEOUT	The specified time has elapsed without the check being properly removed.

See Also **beginInsertion** Method, **endInsertion** Method, **endRemoval** Method.

clearImage Method

Syntax	clearImage (by : <i>int32</i>): void { raises exception, use after open-claim-enable }										
	<table> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><i>by</i></td> <td>Indicates how the image file is to be located so that it can be removed from the storage.</td> </tr> </tbody> </table>	Parameter	Description	<i>by</i>	Indicates how the image file is to be located so that it can be removed from the storage.						
Parameter	Description										
<i>by</i>	Indicates how the image file is to be located so that it can be removed from the storage.										
Remarks	<p>Called to clear a specific image or all the images in the device memory.</p> <p>The following values may be selected for <i>by</i> to initiate clearing of the memory:</p> <table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>CHK_CLR_ALL</td> <td>All images in the device are cleared</td> </tr> <tr> <td>CHK_CLR_BY_FILEID</td> <td>Locate file to be cleared using the FileID property.</td> </tr> <tr> <td>CHK_CLR_BY_FILEINDEX</td> <td>Locate file to be cleared using the FileIndex property.</td> </tr> <tr> <td>CHK_CLR_BY_IMAGETAGDATA</td> <td>Locate file to be cleared using the ImageTagData property.</td> </tr> </tbody> </table>	Value	Meaning	CHK_CLR_ALL	All images in the device are cleared	CHK_CLR_BY_FILEID	Locate file to be cleared using the FileID property.	CHK_CLR_BY_FILEINDEX	Locate file to be cleared using the FileIndex property.	CHK_CLR_BY_IMAGETAGDATA	Locate file to be cleared using the ImageTagData property.
Value	Meaning										
CHK_CLR_ALL	All images in the device are cleared										
CHK_CLR_BY_FILEID	Locate file to be cleared using the FileID property.										
CHK_CLR_BY_FILEINDEX	Locate file to be cleared using the FileIndex property.										
CHK_CLR_BY_IMAGETAGDATA	Locate file to be cleared using the ImageTagData property.										
Return	<p>A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.</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>One of the following errors occurred: <ul style="list-style-type: none"> • Device does not support stored images • Device does not support clearing one image </td> </tr> <tr> <td>E_NOEXIST</td> <td>Image was not found.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	One of the following errors occurred: <ul style="list-style-type: none"> • Device does not support stored images • Device does not support clearing one image 	E_NOEXIST	Image was not found.				
Value	Meaning										
E_ILLEGAL	One of the following errors occurred: <ul style="list-style-type: none"> • Device does not support stored images • Device does not support clearing one image 										
E_NOEXIST	Image was not found.										
See Also	CapStoreImageFiles Property, FileID Property, FileIndex Property, ImageTagData Property.										

defineCropArea Method

Syntax **defineCropArea (cropAreaID: int32, x: int32, y: int32, cx: int32, cy: int32): void { raises exception, use after open-claim-enable }**

Parameter	Description
<i>cropAreaID</i>	The numeric identifier for the defined crop area.
<i>x</i>	The starting X-coordinate of the cropping area.
<i>y</i>	The starting Y-coordinate of the cropping area.
<i>cx</i>	The value added to the “X-coordinate” in order to determine the “X” endpoint for the cropping area.
<i>cy</i>	The value added to the “Y-coordinate” in order to determine the “Y” endpoint for the cropping area.

If the *cropAreaID* parameter is set to `CHK_CROP_AREA_RESET_ALL`, then all the crop area definitions allowed (as specified by the **MaxCropAreas** property) will reset their (*x,y*) and (*cx,cy*) values to (0,0) and (**DocumentWidth**, **DocumentHeight**) respectively.

If the *cropAreaID* parameter is set to `CHK_CROP_AREA_ENTIRE_IMAGE`, then the crop area is equal to the entire area of the scanned image.

If *cx* is set to the parameter `CHK_CROP_AREA_RIGHT`, then the “X” endpoint value will be set to the value of the **DocumentWidth** property.

If *cy* is set to the parameter `CHK_CROP_AREA_BOTTOM`, then the “Y” endpoint value will be set to the value of the **DocumentHeight** property.

Remarks This method is used to establish one or more cropping areas that may be applied to a scanned image. The values are in **MapMode** units and use the top left corner of the scanned document as the origin (0,0). All values are positive.

The **defineCropArea** method specifies an area of interest that is contained within a crop box and given an index number for reference. Only the data defined by **defineCropArea** index number will be sent when the **retrieveImage** method is called.

The crop areas should be set before the **retrieveImage** method is called and will be in effect until changed.

A crop box cannot contain an area larger than that defined by the current **DocumentHeight** and **DocumentWidth** properties. If the resultant value for the endpoint (*x+cx*) is greater than the **DocumentWidth** value, then the “X” endpoint value will be set to **DocumentWidth**. If the resultant value for endpoint (*y+cy*) is greater than the **DocumentHeight** value, then the “Y” endpoint value will be set to **DocumentHeight**.

Errors A `UposException` may be thrown when this method is invoked. For further information, see “Errors” on page 16.

See Also **CapDefineCropArea** Property, **CropAreaCount** Property, **DocumentHeight** Property, **DocumentWidth** Property, **MapMode** Property, **MaxCropAreas** Property.

endInsertion Method

Syntax	endInsertion (): void { raises exception, use after open-claim-enable }						
Remarks	<p>Ends the document insertion processing. If this method call is successful, the device will place the captured image in a working buffer memory area. A StatusUpdateEvent will occur to indicate that a successful scan image process has taken place. No DataEvent is enqueued since data has not been transferred to the ImageData property at this point. The application must invoke retrieveImage in order to populate the ImageData property with the scan image data.</p> <p>When called, the Check Scanner is taken out of the check insertion mode. If a check is not detected in the device, a UposException is raised with an extended error code of ECHK_NOCHECK. This allows an application to prompt the user prior to calling this method to ensure that the form is correctly positioned.</p> <p>This method is paired with the beginInsertion method for controlling check insertion. Although some Check Scanner devices do not require this sort of processing, the application should still use these methods to ensure application portability across different Check Scanner devices.</p>						
Errors	<p>A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.</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>The device is not in check insertion mode.</td> </tr> <tr> <td>E_EXTENDED</td> <td><i>ErrorCodeExtended</i> = ECHK_NOCHECK: The device was taken out of insertion mode without a check being inserted.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	The device is not in check insertion mode.	E_EXTENDED	<i>ErrorCodeExtended</i> = ECHK_NOCHECK: The device was taken out of insertion mode without a check being inserted.
Value	Meaning						
E_ILLEGAL	The device is not in check insertion mode.						
E_EXTENDED	<i>ErrorCodeExtended</i> = ECHK_NOCHECK: The device was taken out of insertion mode without a check being inserted.						
See Also	beginInsertion Method, beginRemoval Method, endRemoval Method, retrieveImage Method.						

endRemoval Method

Syntax	endRemoval (): void { raises exception, use after open-claim-enable }						
Remarks	<p>Ends the document removal processing.</p> <p>When called, the Check Scanner is taken out of check removal or ejection mode. If a check is detected in the device, a UposException is raised with an extended error code of ECHK_CHECK .</p> <p>This method is paired with the beginRemoval method for controlling check removal. Although some Check Scanner devices do not require this sort of processing, the application should still use these methods to ensure application portability across different Check Scanner devices.</p>						
Errors	<p>A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.</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>The device is not in check removal mode.</td> </tr> <tr> <td>E_EXTENDED</td> <td><i>ErrorCodeExtended</i> = ECHK_CHECK: The device was taken out of removal mode while a check is still present.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	The device is not in check removal mode.	E_EXTENDED	<i>ErrorCodeExtended</i> = ECHK_CHECK: The device was taken out of removal mode while a check is still present.
Value	Meaning						
E_ILLEGAL	The device is not in check removal mode.						
E_EXTENDED	<i>ErrorCodeExtended</i> = ECHK_CHECK: The device was taken out of removal mode while a check is still present.						
See Also	beginInsertion Method, beginRemoval Method, endInsertion Method.						

retrieveImage Method

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

Parameter	Description
<i>cropAreaID</i>	Identifier to specify the storage location of the crop area parameters to be applied to the most recently scanned image held in the working area memory of the device. If the value is <code>CHK_CROP_AREA_ENTIRE_IMAGE</code> then the entire area of the most recently scanned image is retrieved.

Remarks Called to retrieve the most recently scanned image which is resident in the work area memory to the **ImageData** property. If this method call is successful, the device will deliver either a **DataEvent** or an **ErrorEvent** at a later time.

If the **CapImageTagData** property is true, then the **ImageTagData** property is set to the tag data associated with the image data file.

If a file name has been created for the image data by the device, then the **FileID** property will be set to the file name; if none is available then the **FileID** property will be set to NULL (0x00).

Many models of Check Scanner devices do not require any check handling processing from the application. Such devices may always be capable of receiving a check, scanning the image into their working memory area, and require no commands to actually read and eject the check. For these type of Check Scanner devices, the **beginInsertion**, **endInsertion**, **beginRemoval** and **endRemoval** methods simply return, and the Control will enqueue the data until the **DataEventEnabled** property is set to true. However, applications should still use these methods to ensure application portability across different Check Scanner devices.

The **retrieveImage** method can not be called after a **retrieveMemory** method has been called until a new document has been scanned.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	The following error has occurred: <ul style="list-style-type: none"> Cropped area that is specified by <i>cropAreaID</i> parameter is invalid.

See Also **CapImageTagData** Property, **FileID** Property, **ImageData** Property, **ImageTagData** Property, **beginInsertion** Method, **beginRemoval** Method, **endInsertion** Method, **endRemoval** Method.

retrieveMemory Method

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

Parameter	Description
<i>by</i>	Indicates how the image file is to be located so that it can be retrieved from the device memory storage.

Remarks Called to retrieve an image that was previously stored in memory to the work area and the **ImageData** property. If this method call is successful, the device will deliver either a **DataEvent** or an **ErrorEvent** at a later time.

The following values may be selected for *by*:

Value	Meaning
CHK_LOCATE_BY_FILEID	Locate image file using the FileID property.
CHK_LOCATE_BY_FILEINDEX	Locate image file using the FileIndex property.
CHK_LOCATE_BY_IMAGETAGDATA	Locate image file using the ImageTagData property.

The **FileID**, **FileIndex**, and **ImageTagData** properties will all be updated to reflect their respective values associated with the image data file after this method is called. A value for **FileIndex** will always be available. The **FileID** and **ImageTagData** properties will be set to NULL (0x00) if the image file does not have respective data to be retrieved for these properties.

The **retrieveImage** method can not be called after a **retrieveMemory** method has been called until a new document has been scanned.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	The following error has occurred: <ul style="list-style-type: none"> • <i>by</i> parameter is invalid. • The image data file could not be located due to an invalid value stored in either the FileID, FileIndex, or ImageTagData properties that was being used with the <i>by</i> value.

See Also **FileID** Property, **FileIndex** Property, **ImageData** Property, **ImageTagData** Property.

storeImage Method

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

Parameter	Description
<i>cropAreaID</i>	Identifier to specify the storage location of the crop area parameters to be applied to image data file currently in the buffer memory area of the device. If the value is <code>CHK_CROP_AREA_ENTIRE_IMAGE</code> , then an exact image of the buffer memory is stored in the device memory (no cropping is applied).

Remarks Called to store an image or a cropped area of the image in the memory of the device.

The **RemainingImagesEstimate** property is adjusted to reflect the approximate number additional images that may be stored in the device memory based upon the file size history of previously stored images.

The **ImageMemoryStatus** property indicates whether or not the device memory is full and is adjusted as a result of this method.

The **FileID**, **FileIndex**, and **ImageTagData** properties must all be updated to reflect their respective values associated with the image data file before this method is called. A value for **FileIndex** will always be available and is supplied by the service. The **FileID** and/or **ImageTagData** properties will be set to NULL (0x00) if the device does not support the respective property.

Return A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_EXIST	Image already exists in the store location specified by the FileIndex property.
E_ILLEGAL	One of the following errors occurred: <ul style="list-style-type: none"> • Device does not support storing images • Cropped area that is specified by <i>cropAreaID</i> parameter is invalid.
E_FAILURE	Internal error storing image.
E_EXTENDED	<i>ErrorCodeExtended</i> = ECHK_NOROOM: There is no more room for the image in memory.

See Also **CapStoreImageFiles** Property, **FileID** Property, **FileIndex** Property, **ImageMemoryStatus** Property, **ImageTagData** Property, **RemainingImagesEstimate** Property.

Events (UML interfaces)

DataEvent

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

Description Notifies the application when data from the Check Scanner device is available to be read.

Attributes This event contains the following attribute:

Attributes	Type	Description
<i>Status</i>	<i>int32</i>	Set to 0.

Remarks Before this event is delivered, the scanned check image is placed into **ImageData**.

See Also **ImageData** Property, **endInsertion** Method, **retrieveImage** Method, **storeImage** Method.

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

See Also "Events" on page 15, **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 Check Scanner device and a suitable response by the application is necessary to process the error condition.

Attributes This event contains the following attributes:

Attributes	Type	Description
<i>ErrorCode</i>	<i>int32</i>	Error code causing the error event. See a list of Error Codes on page 16.
<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_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 This event is not delivered until **DataEventEnabled** is true and other event delivery requirements are met, so that proper application sequencing occurs.

See Also “Device Input Model” on page 18, “Device States” on page 26.

StatusUpdateEvent

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

Description Notifies the application that there is a change in the status of the Check Scanner 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 Check Scanner device.

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

Value	Meaning
CHK_SUE_SCANCOMPLETE	The process of scanning a document image has been successfully completed. <i>Note that Release 1.3</i> added Power State Reporting with additional <i>Power reporting StatusUpdateEvent values</i> . See “StatusUpdateEvent” description on page 63.

Remarks Enqueued after the **endInsertion** method has been called and the Check Scanner device has successfully completed the process of scanning a new image into a working buffer memory area. Also enqueued when the Check Scanner device detects a power state change.

See Also “Events” on page 15.

Coin Dispenser

This Chapter defines the Coin Dispenser device category.

Summary

Properties (UML attributes)

<i>Common</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</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
CapStatisticsReporting:	<i>boolean</i>	{ read-only }	1.8	open
CapUpdateStatistics:	<i>boolean</i>	{ read-only }	1.8	open
CheckHealthText:	<i>string</i>	{ read-only }	1.0	open
Claimed:	<i>boolean</i>	{ read-only }	1.0	open
DataCount:	<i>int32</i>	{ read-only }	1.2	Not Supported
DataEventEnabled:	<i>boolean</i>	{ read-write }	1.0	Not Supported
DeviceEnabled:	<i>boolean</i>	{ read-write }	1.0	open & claim
FreezeEvents:	<i>boolean</i>	{ read-write }	1.0	open
OutputID:	<i>int32</i>	{ read-only }	1.0	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 }	1.0	--
DeviceControlDescription:	<i>string</i>	{ read-only }	1.0	--
DeviceControlVersion:	<i>int32</i>	{ read-only }	1.0	--
DeviceServiceDescription:	<i>string</i>	{ read-only }	1.0	open
DeviceServiceVersion:	<i>int32</i>	{ read-only }	1.0	open
PhysicalDeviceDescription:	<i>string</i>	{ read-only }	1.0	open
PhysicalDeviceName:	<i>string</i>	{ read-only }	1.0	open

Properties (Continued)

<i>Specific</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
CapEmptySensor:	boolean	{ read-only }	1.0	open
CapJamSensor:	boolean	{ read-only }	1.0	open
CapNearEmptySensor:	boolean	{ read-only }	1.0	open
DispenserStatus:	int32	{ read-only }	1.0	open, claim, & enable

Methods (UML operations)

Common

<i>Name</i>	<i>Version</i>
open (logicalDeviceName: <i>string</i>): void { raises-exception }	1.0
close (): void { raises-exception, use after open }	1.0
claim (timeout: <i>int32</i>): void { raises-exception, use after open }	1.0
release (): void { raises-exception, use after open, claim }	1.0
checkHealth (level: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.0
clearInput (): void { }	<i>Not supported</i>
clearOutput (): void { }	<i>Not supported</i>
directIO (command: <i>int32</i> , inout data: <i>int32</i> , inout obj: <i>object</i>): void { raises-exception, use after open }	1.0
resetStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
retrieveStatistics (inout statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
updateStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8

Specific

<i>Name</i>	<i>Version</i>
dispenseChange (amount: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.0

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>
upos::events::DataEvent		<i>Not Supported</i>	
upos::events::DirectIOEvent			1.0
EventNumber:	<i>int32</i>	{ read-only }	
Data:	<i>int32</i>	{ read-write }	
Obj:	<i>object</i>	{ read-write }	
upos::events::ErrorEvent		<i>Not Supported</i>	
upos::events::OutputCompleteEvent		<i>Not Supported</i>	
upos::events::StatusUpdateEvent			
Status:	<i>int32</i>	{ read-only }	1.0

General Information

The Coin Dispenser programmatic name is “CoinDispenser”.

Capabilities

The coin dispenser has the following capability:

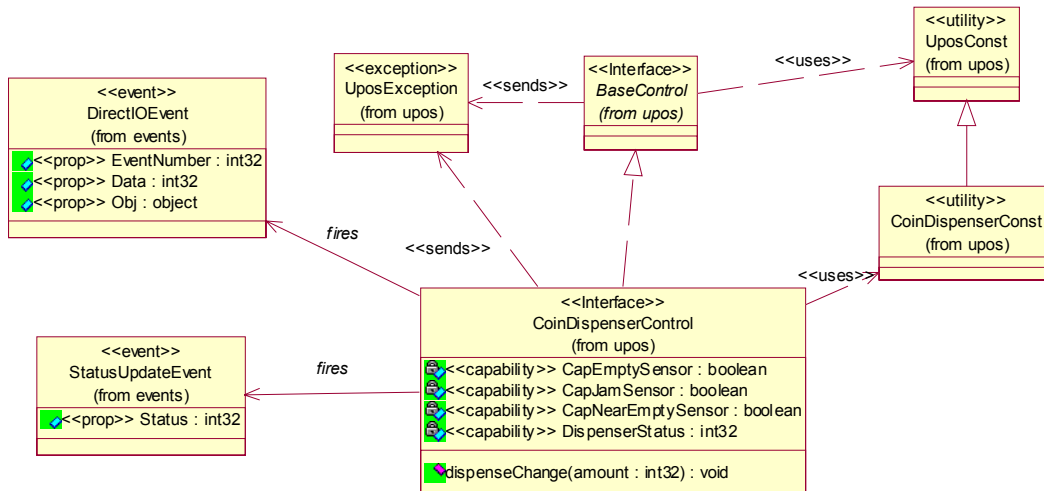
- Supports a method that allows a specified amount of change to be dispensed from the device.

The coin dispenser may have the following additional capability:

- Status reporting, which indicates empty coin slot conditions, near empty coin slot conditions, and coin slot jamming conditions.

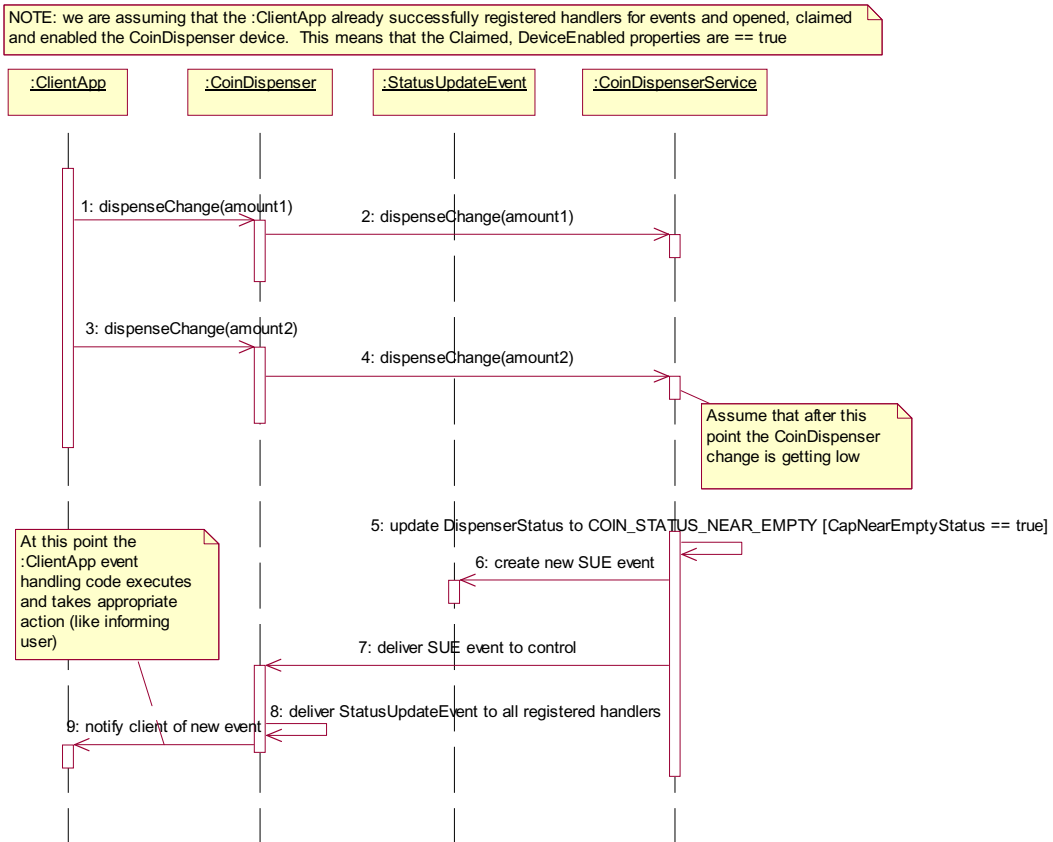
Coin Dispenser Class Diagram

The following diagram shows the relationships between the Coin Dispenser classes.



Coin Dispenser Sequence Diagram *Added in Release 1.7*

The following sequence diagram shows the typical usage of the Coin Dispenser device, showing coin dispensing and the firing of a **StatusUpdateEvent** due to coin status getting low.



Model

The general model of a coin dispenser is:

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

Device Sharing

The coin dispenser is an exclusive-use device, as follows:

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

Properties (UML attributes)

CapEmptySensor Property

Syntax	CapEmptySensor: <i>boolean</i> { read-only, access after open }
Remarks	If true, the coin dispenser can report an out-of-coinage condition. 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 16.

CapJamSensor Property

Syntax	CapJamSensor: <i>boolean</i> { read-only, access after open }
Remarks	If true, the coin dispenser can report a mechanical jam or failure condition. 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 16.

CapNearEmptySensor Property

Syntax	CapNearEmptySensor: <i>boolean</i> { read-only, access after open }
Remarks	If true, the coin dispenser can report when it is almost out of coinage. 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 16.

DispenserStatus Property

Syntax	DispenserStatus: <i>int32</i> { read-only, access after open-claim-enable }										
Remarks	Holds the current status of the dispenser. It has one of the following values: <table border="1" data-bbox="467 1386 1380 1764"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>COIN_STATUS_OK</td> <td>Ready to dispense coinage. This value is also set when the dispenser is unable to detect an error condition.</td> </tr> <tr> <td>COIN_STATUS_EMPTY</td> <td>Cannot dispense coinage because the dispenser is empty.</td> </tr> <tr> <td>COIN_STATUS_NEAREMPTY</td> <td>Can still dispense coinage, but the dispenser is nearly empty.</td> </tr> <tr> <td>COIN_STATUS_JAM</td> <td>A mechanical fault has occurred.</td> </tr> </tbody> </table> This property is initialized and kept current while the device is enabled.	Value	Meaning	COIN_STATUS_OK	Ready to dispense coinage. This value is also set when the dispenser is unable to detect an error condition.	COIN_STATUS_EMPTY	Cannot dispense coinage because the dispenser is empty.	COIN_STATUS_NEAREMPTY	Can still dispense coinage, but the dispenser is nearly empty.	COIN_STATUS_JAM	A mechanical fault has occurred.
Value	Meaning										
COIN_STATUS_OK	Ready to dispense coinage. This value is also set when the dispenser is unable to detect an error condition.										
COIN_STATUS_EMPTY	Cannot dispense coinage because the dispenser is empty.										
COIN_STATUS_NEAREMPTY	Can still dispense coinage, but the dispenser is nearly empty.										
COIN_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 16.										

Methods (UML operations)

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.

Remarks Dispenses change. The value represented by the *amount* parameter is a count of the currency units to dispense (such as cents or yen).

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	An <i>amount</i> parameter value of zero was specified, or the <i>amount</i> parameter contained a negative value or a value greater than the device can dispense.

Events (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 Coin Dispenser 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 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 Coin Dispenser devices which may not have any knowledge of the Service's need for this event.

See Also "Events" on page 15, **directIO** Method.

StatusUpdateEvent

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

Description Notifies the application of a sensor status change.

Attributes This event contains the following attribute:

Attribute	Type	Description
<i>Status</i>	<i>int32</i>	The status reported from the Coin Dispenser.

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

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

Note that Release 1.3 added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See “StatusUpdateEvent” description on page 63.

Remarks This event applies for status changes of the sensor types supported, as indicated by the capability properties. It also applies if Power State Reporting is enabled.

See Also “Events” on page 15.

Fiscal Printer

This Chapter defines the Fiscal Printer device category.

Summary

Properties (UML attributes)

<i>Common</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</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
CapStatisticsReporting:	<i>boolean</i>	{ read-only }	1.8	open
CapUpdateStatistics:	<i>boolean</i>	{ read-only }	1.8	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

Properties (Continued)

<i>Specific</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
CapAdditionalHeader:	<i>boolean</i>	{ read-only }	1.6	open
CapAdditionalLines:	<i>boolean</i>	{ read-only }	1.3	open
CapAdditionalTrailer:	<i>boolean</i>	{ read-only }	1.6	open
CapAmountAdjustment:	<i>boolean</i>	{ read-only }	1.3	open
CapAmountNotPaid:	<i>boolean</i>	{ read-only }	1.3	open
CapChangeDue:	<i>boolean</i>	{ read-only }	1.6	open
CapCheckTotal:	<i>boolean</i>	{ read-only }	1.3	open
CapCoverSensor: (1)	<i>boolean</i>	{ read-only }	1.3	open
CapDoubleWidth:	<i>boolean</i>	{ read-only }	1.3	open
CapDuplicateReceipt:	<i>boolean</i>	{ read-only }	1.3	open
CapEmptyReceiptIsVoidable:	<i>boolean</i>	{ read-only }	1.6	open
CapFiscalReceiptStation:	<i>boolean</i>	{ read-only }	1.6	open
CapFiscalReceiptType:	<i>boolean</i>	{ read-only }	1.6	open
CapFixedOutput:	<i>boolean</i>	{ read-only }	1.3	open
CapHasVatTable:	<i>boolean</i>	{ read-only }	1.3	open
CapIndependentHeader:	<i>boolean</i>	{ read-only }	1.3	open
CapItemList:	<i>boolean</i>	{ read-only }	1.3	open
CapJrnEmptySensor: (1)	<i>boolean</i>	{ read-only }	1.3	open
CapJrnNearEndSensor: (1)	<i>boolean</i>	{ read-only }	1.3	open
CapJrnPresent: (1)	<i>boolean</i>	{ read-only }	1.3	open
CapMultiContractor:	<i>boolean</i>	{ read-only }	1.6	open
CapNonFiscalMode:	<i>boolean</i>	{ read-only }	1.3	open
CapOnlyVoidLastItem:	<i>boolean</i>	{ read-only }	1.6	open
CapOrderAdjustmentFirst:	<i>boolean</i>	{ read-only }	1.3	open
CapPackageAdjustment:	<i>boolean</i>	{ read-only }	1.6	open
CapPercentAdjustment:	<i>boolean</i>	{ read-only }	1.3	open
CapPositiveAdjustment:	<i>boolean</i>	{ read-only }	1.3	open
CapPostPreLine:	<i>boolean</i>	{ read-only }	1.6	open
CapPowerLossReport:	<i>boolean</i>	{ read-only }	1.3	open
CapPredefinedPaymentLines:	<i>boolean</i>	{ read-only }	1.3	open
CapReceiptNotPaid:	<i>boolean</i>	{ read-only }	1.3	open
CapRecEmptySensor: (1)	<i>boolean</i>	{ read-only }	1.3	open
CapRecNearEndSensor: (1)	<i>boolean</i>	{ read-only }	1.3	open
CapRecPresent: (1)	<i>boolean</i>	{ read-only }	1.3	open

Properties (Continued)

<i>Specific (continued)</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
CapRemainingFiscalMemory:	<i>boolean</i>	{ read-only }	1.3	open
CapReservedWord:	<i>boolean</i>	{ read-only }	1.3	open
CapSetCurrency:	<i>boolean</i>	{ read-only }	1.6	open
CapSetHeader:	<i>boolean</i>	{ read-only }	1.3	open
CapSetPOSID:	<i>boolean</i>	{ read-only }	1.3	open
CapSetStoreFiscalID:	<i>boolean</i>	{ read-only }	1.3	open
CapSetTrailer:	<i>boolean</i>	{ read-only }	1.3	open
CapSetVatTable:	<i>boolean</i>	{ read-only }	1.3	open
CapSlpEmptySensor: (1)	<i>boolean</i>	{ read-only }	1.3	open
CapSlpFiscalDocument:	<i>boolean</i>	{ read-only }	1.3	open
CapSlpFullSlip: (1)	<i>boolean</i>	{ read-only }	1.3	open
CapSlpNearEndSensor: (1)	<i>boolean</i>	{ read-only }	1.3	open
CapSlpPresent: (1)	<i>boolean</i>	{ read-only }	1.3	open
CapSlpValidation:	<i>boolean</i>	{ read-only }	1.3	open
CapSubAmountAdjustment:	<i>boolean</i>	{ read-only }	1.3	open
CapSubPercentAdjustment:	<i>boolean</i>	{ read-only }	1.3	open
CapSubtotal:	<i>boolean</i>	{ read-only }	1.3	open
CapTotalizerType:	<i>boolean</i>	{ read-only }	1.6	open
CapTrainingMode:	<i>boolean</i>	{ read-only }	1.3	open
CapValidateJournal:	<i>boolean</i>	{ read-only }	1.3	open
CapXReport:	<i>boolean</i>	{ read-only }	1.3	open
ActualCurrency:	<i>int32</i>	{ read-only }	1.6	open, claim, & enable
AdditionalHeader:	<i>string</i>	{ read-write }	1.6	open, claim, & enable
AdditionalTrailer:	<i>string</i>	{ read-write }	1.6	open, claim, & enable
AmountDecimalPlaces:	<i>int32</i>	{ read-only }	1.3	open, claim, & enable
AsyncMode:	<i>boolean</i>	{ read-write }	1.3	open
ChangeDue:	<i>string</i>	{ read-write }	1.6	open
CheckTotal:	<i>boolean</i>	{ read-write }	1.3	open
ContractorId:	<i>int32</i>	{ read-write }	1.6	open, claim, & enable
CountryCode:	<i>int32</i>	{ read-only }	1.3	open, claim, & enable
CoverOpen: (1)	<i>boolean</i>	{ read-only }	1.3	open, claim, & enable
DateType:	<i>int32</i>	{ read-write }	1.6	open, claim, & enable
DayOpened:	<i>boolean</i>	{ read-only }	1.3	open, claim, & enable
DescriptionLength:	<i>int32</i>	{ read-only }	1.3	open
DuplicateReceipt:	<i>boolean</i>	{ read-write }	1.3	open
ErrorLevel:	<i>int32</i>	{ read-only }	1.3	open

Properties (Continued)

<i>Specific (continued)</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
ErrorOutID:	<i>int32</i>	{ read-only }	1.3	open, claim, & enable
ErrorState:	<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
FiscalReceiptStation:	<i>int32</i>	{ read-write }	1.6	open, claim, & enable
FiscalReceiptType:	<i>int32</i>	{ read-write }	1.6	open, claim, & enable
FlagWhenIdle: (1)	<i>boolean</i>	{ read-write }	1.3	open
JrnEmpty:	<i>boolean</i>	{ read-only }	1.3	open, claim, & enable
JrnNearEnd:	<i>boolean</i>	{ read-only }	1.3	open, claim, & enable
MessageLength:	<i>int32</i>	{ read-only }	1.3	open
MessageType:	<i>int32</i>	{ read-write }	1.6	open
NumHeaderLines:	<i>int32</i>	{ read-only }	1.3	open
NumTrailerLines:	<i>int32</i>	{ read-only }	1.3	open
NumVatRates:	<i>int32</i>	{ read-only }	1.3	open
PostLine:	<i>string</i>	{ read-write }	1.6	open, claim, & enable
PredefinedPaymentLines:	<i>string</i>	{ read-only }	1.3	open
PreLine:	<i>string</i>	{ read-write }	1.6	open, claim, & enable
PrinterState:	<i>int32</i>	{ read-only }	1.3	open, claim, & enable
QuantityDecimalPlaces:	<i>int32</i>	{ read-only }	1.3	open, claim, & enable
QuantityLength:	<i>int32</i>	{ read-only }	1.3	open, claim, & enable
RecEmpty: (1)	<i>boolean</i>	{ read-only }	1.3	open, claim, & enable
RecNearEnd: (1)	<i>boolean</i>	{ read-only }	1.3	open, claim, & enable
RemainingFiscalMemory:	<i>int32</i>	{ read-only }	1.3	open, claim, & enable
ReservedWord:	<i>string</i>	{ read-only }	1.3	open
SlpEmpty: (1)	<i>boolean</i>	{ read-only }	1.3	open, claim, & enable
SlpNearEnd: (1)	<i>boolean</i>	{ read-only }	1.3	open, claim, & enable
SlipSelection:	<i>int32</i>	{ read-write }	1.3	open, claim, & enable
TotalizerType:	<i>int32</i>	{ read-write }	1.6	open, claim, & enable
TrainingModeActive:	<i>boolean</i>	{ read-only }	1.3	open, claim, & enable

Methods (UML operations)**Common**

<i>Name</i>	<i>Version</i>
open (logicalDeviceName: <i>string</i>): void { raises-exception }	1.3
close (): void { raises-exception, use after open }	1.3
claim (timeout: <i>int32</i>): void { raises-exception, use after open }	1.3
release (): void { raises-exception, use after open, claim }	1.3
checkHealth (level: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3
clearInput (): void { }	<i>Not supported</i>
clearOutput (): void { raises-exception, use after open, claim }	1.3
directIO (command: <i>int32</i>, inout data: <i>int32</i>, inout obj: <i>object</i>): void { raises-exception, use after open }	1.3
resetStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
retrieveStatistics (inout statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
updateStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8

Specific - Presetting Fiscal

setCurrency (newCurrency: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.6
setDate (date: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.3
setHeaderLine (lineNumber: <i>int32</i>, text: <i>string</i>, doubleWidth: <i>boolean</i>): void { raises-exception, use after open, claim, enable }	1.3
setPOSID (POSID: <i>string</i>, cashierID: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.3
setStoreFiscalID (ID: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.3
setTrailerLine (lineNumber: <i>int32</i>, text: <i>string</i>, doubleWidth: <i>boolean</i>): void { raises-exception, use after open, claim, enable }	1.3
setVatTable (): void { raises-exception, use after open, claim, enable }	1.3
setVatValue (vatID: <i>int32</i>, vatValue: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.3

Specific - Fiscal Receipt

beginFiscalReceipt (printHeader: <i>boolean</i>): void { raises-exception, use after open, claim, enable }	1.3
endFiscalReceipt (printHeader: <i>boolean</i>): void { raises-exception, use after open, claim, enable }	1.3
printDuplicateReceipt (): void { raises-exception, use after open, claim, enable }	1.3
printRecCash (amount: <i>currency</i>): void { raises-exception, use after open, claim, enable }	1.6
printRecItem (description: <i>string</i>, price: <i>currency</i>, quantity: <i>int32</i>, vatInfo: <i>int32</i>, unitPrice: <i>currency</i>, unitName: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.3
printRecItemAdjustment (adjustmentType: <i>int32</i>, description: <i>string</i>, amount: <i>currency</i>, vatInfo: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3
printRecItemFuel (description: <i>string</i>, price: <i>currency</i>, quantity: <i>int32</i>, vatInfo: <i>int32</i>, unitPrice: <i>currency</i>, unitName: <i>string</i>, specialTax: <i>currency</i>, specialTaxName: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.6
printRecItemFuelVoid (description: <i>string</i>, price: <i>currency</i>, vatInfo: <i>int32</i>, specialTax: <i>currency</i>): void { raises-exception, use after open, claim, enable }	1.6
printRecMessage (message: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.3
printRecNotPaid (description: <i>string</i>, amount: <i>currency</i>): void { raises-exception, use after open, claim, enable }	1.3
printRecPackageAdjustment (adjustmentType: <i>int32</i>, description: <i>string</i>, vatAdjustment: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.6
printRecPackageAdjustVoid (adjustmentType: <i>int32</i>, vatAdjustment: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.6
printRecRefund (description: <i>string</i>, amount: <i>currency</i>, vatInfo: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3
printRecRefundVoid (description: <i>string</i>, amount: <i>currency</i>, vatInfo: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.6
printRecSubtotal (amount: <i>currency</i>): void { raises-exception, use after open, claim, enable }	1.3
printRecSubtotalAdjustment (adjustmentType: <i>int32</i>, description: <i>string</i>, amount: <i>currency</i>): void { raises-exception, use after open, claim, enable }	1.3
printRecSubtotalAdjustVoid (adjustmentType: <i>int32</i>, amount: <i>currency</i>): void { raises-exception, use after open, claim, enable }	1.6
printRecTaxID (taxId: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.6
printRecTotal (total: <i>currency</i>, payment: <i>currency</i>, description: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.3
printRecVoid (description: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.3

printRecVoidItem (*description: string, amount: currency, quantity: int32, adjustmentType: int32, adjustment: currency, vatInfo: int32*): 1.3
void { raises-exception, use after open, claim, enable }

Specific - Fiscal Document

beginFiscalDocument (*documentAmount: int32*): 1.3
void { raises-exception, use after open, claim, enable }

endFiscalDocument (): 1.3
void { raises-exception, use after open, claim, enable }

printFiscalDocumentLine (*documentLine: string*): 1.3
void { raises-exception, use after open, claim, enable }

Specific - Item Lists

beginItemList (*vatID: int32*): 1.3
void { raises-exception, use after open, claim, enable }

endItemList (): 1.3
void { raises-exception, use after open, claim, enable }

verifyItem (*itemName: string, vatID: int32*): 1.3
void { raises-exception, use after open, claim, enable }

Specific - Fiscal Reports

printPeriodicTotalsReport (*date1: string, date2: string*): 1.3
void { raises-exception, use after open, claim, enable }

printPowerLossReport (): 1.3
void { raises-exception, use after open, claim, enable }

printReport (*reportType: int32, startNum: string, endNum: string*): 1.3
void { raises-exception, use after open, claim, enable }

printXReport (): 1.3
void { raises-exception, use after open, claim, enable }

printZReport (): 1.3
void { raises-exception, use after open, claim, enable }

Specific - Slip Insertion

beginInsertion (*timeout: int32*): 1.3
void { raises-exception, use after open, claim, enable } (1)

beginRemoval (*timeout: int32*): 1.3
void { raises-exception, use after open, claim, enable } (1)

endInsertion (): 1.3
void { raises-exception, use after open, claim, enable } (1)

endRemoval (): 1.3
void { raises-exception, use after open, claim, enable } (1)

Specific - Non-Fiscal

beginFixedOutput (station: <i>int32</i>, documentType: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3
beginNonFiscal (): void { raises-exception, use after open, claim, enable }	1.3
beginTraining (): void { raises-exception, use after open, claim, enable }	1.3
endFixedOutput (): void { raises-exception, use after open, claim, enable }	1.3
endNonFiscal (): void { raises-exception, use after open, claim, enable }	1.3
endTraining (): void { raises-exception, use after open, claim, enable }	1.3
printFixedOutput (documentType: <i>int32</i>, lineNumber: <i>int32</i>, data: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.3
printNormal (station: <i>int32</i>, data: <i>string</i>): void { raises-exception, use after open, claim, enable } (1)	1.3

Specific - Data Requests

getData (dataItem: <i>int32</i>, inout optArgs: <i>int32</i>, inout data: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.3
getDate (inout date: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.3
getTotalizer (vatID: <i>int32</i>, optArgs: <i>int32</i>, inout data: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.3
getVatEntry (vatID: <i>int32</i>, optArgs: <i>int32</i>, inout vatRate: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3

Specific - Error Corrections

clearError (): void { raises-exception, use after open, claim, enable }	1.3
resetPrinter (): void { raises-exception, use after open, claim, enable }	1.3

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>
upos::events::DataEvent		<i>Not Supported</i>	
upos::events::DirectIOEvent			1.3
EventNumber:	<i>int32</i>	{ read-only }	
Data:	<i>int32</i>	{ read-write }	
Obj:	<i>object</i>	{ read-write }	
upos::events::ErrorEvent			1.3
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			1.3
OutputID:	<i>int32</i>	{ read-only }	
upos::events::StatusUpdateEvent			1.3
Status:	<i>int32</i>	{ read-only }	

Note:

(1) Properties and methods marked with (1) are adapted from the POS Printer device.

General Information

The Fiscal Printer programmatic name is “FiscalPrinter”.

The Fiscal Printer Control does not attempt to encapsulate a generic graphics printer. Rather, for performance and ease of use considerations, the interfaces are defined to directly control the normal printer functions.

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

- Brazil
- Bulgaria
- Greece
- Hungary
- Italy
- Poland
- Romania
- Russia
- Turkey

The Fiscal Printer model defines three stations with the following general uses:

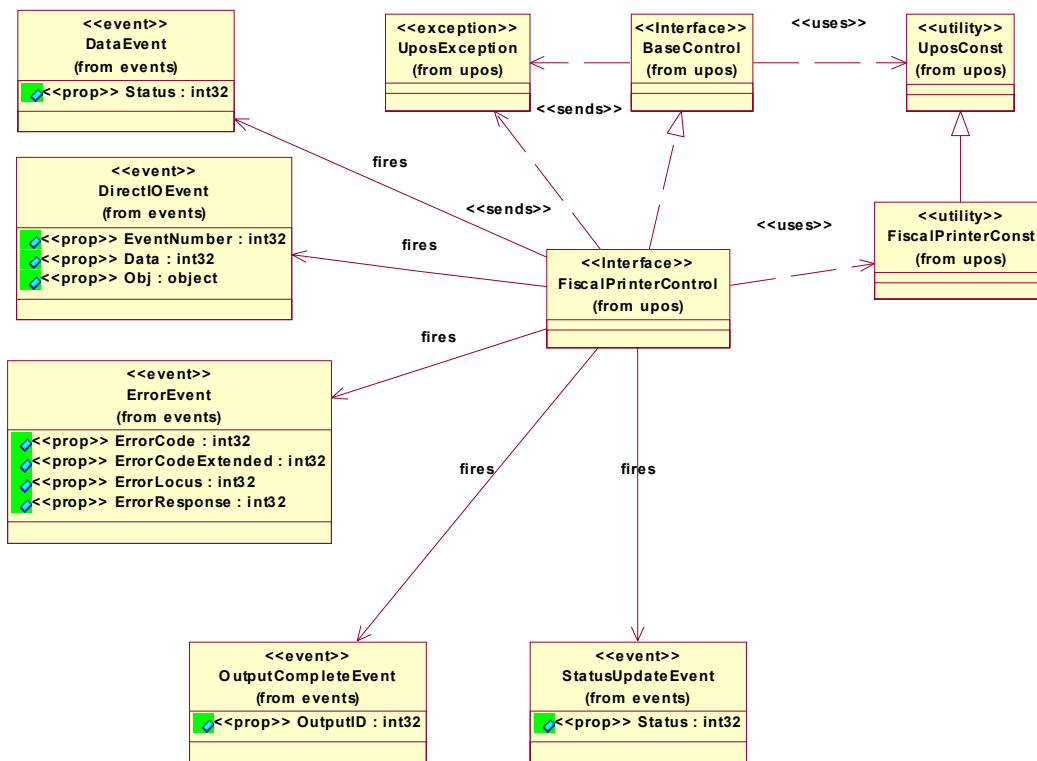
- **Journal** Used for simple text to log transaction and activity information. Kept by the store for audit and other purposes.
- **Receipt** Used to print transaction information. It is mandatory to give a printed fiscal receipt to the customer. Also often used for store reports. Contains either a knife to cut the paper between transactions, or a tear bar to manually cut the paper.
- **Slip** Used to print information on a form. Usually given to the customer.
The **Slip** station is also used to print “validation” information on a form. The form type is typically a check or credit card slip.
It may also be used to print complete transaction information instead of printing it on the receipt station.

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

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

Fiscal Printer Class Diagram

The following diagram shows the relationships between the Fiscal Printer classes.



General Requirements

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

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

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

Fiscal Printer Modes

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

There are three possible Fiscal Printer modes:

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

Model

Updated in Release 1.8

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

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

printFiscalDocumentLine
printFixedOutput
printNormal
printRecCash
printRecItem
printRecItemAdjustment
printRecItemFuel
printRecItemFuelVoid
printRecMessage
printRecNotPaid
printRecPackageAdjustment
printRecPackageAdjustVoid
printRecRefund
printRecRefundVoid
printRecSubtotal
printRecSubtotalAdjustment
printRecSubtotalAdjustVoid
printRecTaxID
printRecTotal
printRecVoid
printRecVoidItem

When `AsyncMode` is false, then these methods print synchronously.

When `AsyncMode` is true, then these methods operate as follows:

- The Device buffers the request in program memory, for delivery to the Physical Device as soon as the Physical Device can receive and process it, sets the `OutputID` property to an identifier for this request, and returns as soon as possible. When the device completes the request successfully, the `OutputCompleteEvent` is enqueued. A parameter of this event contains the `OutputID` of the completed request.

Asynchronous Fiscal Printer methods will not throw a `UposException` due to a printing problem, such as out of paper or Fiscal Printer fault. These errors will only be reported by an `ErrorEvent`. A `UposException` is thrown only if the Fiscal Printer is not claimed and enabled, a parameter is invalid, or the request cannot be enqueued. The first two error cases are due to an application error, while the last is a serious system resource exception.

- If an error occurs while performing an asynchronous request, an `ErrorEvent` is enqueued. The `ErrorStation` property is set to the station or stations that were printing when the error occurred. The `ErrorLevel`, `ErrorString` and `ErrorState` and `ErrorOutID` properties are also set.

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

- Asynchronous output is performed on a first-in first-out basis.
- All buffered output data, including all asynchronous output, 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.

Error Model

The Fiscal Printer error reporting model is as follows:

- Most of the Fiscal Printer error conditions are reported by setting the *UpoxException*'s (or *ErrorEvent*'s) *ErrorCode* to *E_EXTENDED* and then setting *ErrorCodeExtended* to one of the following:

EFPTR_COVER_OPEN

The Fiscal Printer cover is open.

EFPTR_JRN_EMPTY

The journal station has run out of paper.

EFPTR_REC_EMPTY

The receipt station has run out of paper.

EFPTR_SLP_EMPTY

The slip station has run out of paper.

EFPTR_MISSING_DEVICES

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

EFPTR_WRONG_STATE

The requested method could not be executed in the Fiscal Printer's current state.

EFPTR_TECHNICAL_ASSISTANCE

The Fiscal Printer has encountered a severe error condition. Calling for Fiscal Printer technical assistance is required.

EFPTR_CLOCK_ERROR

The Fiscal Printer's internal clock has failed.

EFPTR_FISCAL_MEMORY_FULL

The Fiscal Printer's fiscal memory has been exhausted.

EFPTR_FISCAL_MEMORY_DISCONNECTED

The Fiscal Printer's fiscal memory has been disconnected.

EFPTR_FISCAL_TOTALS_ERROR

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

EFPTR_BAD_ITEM_QUANTITY

The quantity parameter is invalid.

EFPTR_BAD_ITEM_AMOUNT

The amount parameter is invalid.

EFPTR_BAD_ITEM_DESCRIPTION

The description parameter is either too long, contains illegal characters or contains a reserved word.

EFPTR_RECEIPT_TOTAL_OVERFLOW

The receipt total has overflowed.

EFPTR_BAD_VAT

The vat parameter is invalid.

EFPTR_BAD_PRICE

The price parameter is invalid.

EFPTR_BAD_DATE

The date parameter is invalid.

EFPTR_NEGATIVE_TOTAL

The Fiscal Printer's computed total or subtotal is less than zero.

EFPTR_WORD_NOT_ALLOWED

The description contains the reserved word.

EFPTR_BAD_LENGTH

The length of the string to be printed as post or pre line is too long.

EFPTR_MISSING_SET_CURRENCY

The Fiscal Printer is expecting the activation of a new currency.

Other Fiscal Printer errors are reported by setting the exception's (or **ErrorEvent**'s) *ErrorCode* to E_FAILURE or another error status. These failures are typically due to a Fiscal Printer fault or jam, or to a more serious error.

Release 1.8 additional Model clarifications

While the Fiscal Printer is enabled, the printer state is monitored, and changes are reported to the application. Most Fiscal Printer statuses are reported by both firing a **StatusUpdateEvent** and by updating a printer property. Statuses, as defined in the later properties and events sections, are:

Prior to Release 1.8

StatusUpdateEvent	Property
FPTR_SUE_COVER_OPEN	CoverOpen = true
FPTR_SUE_COVER_OK	CoverOpen = false
FPTR_SUE_JRN_EMPTY	JrnEmpty = true
FPTR_SUE_JRN_NEAREMPTY	JrnNearEnd = true
FPTR_SUE_JRN_PAPEROK	JrnEmpty = JrnNearEnd = false
FPTR_SUE_REC_EMPTY	RecEmpty = true
FPTR_SUE_REC_NEAREMPTY	RecNearEnd = true
FPTR_SUE_REC_PAPEROK	RecEmpty = RecNearEnd = false
FPTR_SUE_SLP_EMPTY	SlpEmpty = true
FPTR_SUE_SLP_NEAREMPTY	SlpNearEnd = true
FPTR_SUE_SLP_PAPEROK	SlpEmpty = SlpNearEnd = false

Release 1.8 and later

FPTR_SUE_JRN_COVER_OPEN	CoverOpen = true
FPTR_SUE_JRN_COVER_OK	CoverOpen = false if all covers closed; CoverOpen = true if any other cover is open
FPTR_SUE_REC_COVER_OPEN	CoverOpen = true
FPTR_SUE_REC_COVER_OK	CoverOpen = false if all covers closed; CoverOpen = true if any other cover is open
FPTR_SUE_SLP_COVER_OPEN	CoverOpen = true
FPTR_SUE_SLP_COVER_OK	CoverOpen = false if all covers closed; CoverOpen = true if any other cover is open

Release 1.8 – Clarification

The Fiscal Printer’s slip station statuses must be reported independently from the slip insertion and removal methods – **beginInsertion** / **endInsertion** and **beginRemoval** / **endRemoval**. This is important because some applications base logic decisions upon Fiscal Printer state changes. That is, the application will only perform slip insertion after knowing that a slip has been placed at the entrance to the slip station. An example: After the Total key is pressed, the application enters tendering mode. It begins to monitor peripherals and the keyboard to determine the type of tender to perform. If a credit or debit card is swiped at an MSR, then its **DataEvent** causes the application to begin credit/debit tender. But if a form is placed at the slip station, then its **StatusUpdateEvent** or **SlpEmpty** property change causes the application to begin a check MICR read.

When a form is placed at the entrance to the slip station, the Fiscal Printer must fire a PTR_SUE_SLP_PAPEROK **StatusUpdateEvent** and set the **SlpEmpty** and **SlpNearEnd** properties to false. The application may then call the **beginInsertion** and **endInsertion** methods with reasonable confidence that they will succeed. Note that it must not be assumed that the form is ready for printing after the PTR_SUE_SLP_PAPEROK is received. Only after successful **beginInsertion** and **endInsertion** calls is the form ready for printing.

When a form is removed from the slip station, the Fiscal Printer must fire a PTR_SUE_SLP_EMPTY **StatusUpdateEvent** and set the **SlpEmpty** property to true. If the **beginInsertion** and **endInsertion** method sequence has not been called, then removing the form from the slip station entrance will cause this to occur. If this method sequence has successfully completed, then the event and property change will typically occur after a **beginRemoval** and **endRemoval** method sequence. But they would also occur if the slip prints beyond the end of the form or if the form is forcibly removed.

Exception: The design of some Fiscal Printers makes it impossible for a service to determine the presence of a form until the printer “jaws” are opened, which occurs when **beginInsertion** is called. This exception is largely limited to cases where the **CapSlpFullslip** property is false, indicating a “validation” type of slip station. Validation stations typically use the same Fiscal Printer mechanism as the receipt and/or journal stations. In these cases, the slip status events must be fired as soon as possible, given the constraints of the device.

Fiscal Printer States

Updated in Release 1.8

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

The current state of the Fiscal Printer is kept in the **PrinterState** property.

The Fiscal Printer has the following states:

- **Monitor:**
This is a neutral state. From this state, it is possible to move to most of the other Fiscal Printer states. After a successful call to the **claim** method and successful setting of the **DeviceEnabled** property to true the Fiscal Printer should be in this state unless there is a Fiscal Printer error.
- **Fiscal Receipt:**
The Fiscal Printer is processing a fiscal receipt. All **printRec...** methods except **printRecNotPaid** and **printRecTaxID** are available for use while in this state. This state is entered from the **Monitor** state using the **beginFiscalReceipt** method.
- **Fiscal Receipt Total:**
The Fiscal Printer has already accepted at least one payment method, but the receipt's total amount has not yet been tendered. This state is entered from the **Fiscal Receipt** state by use of the **printRecTotal** method. The Fiscal Printer remains in this state while the total remains unpaid. This state can be left by using the **printRecTotal**, **printRecNotPaid** or **printRecVoid** methods.
- **Fiscal Receipt Ending:**
The Fiscal Printer has completed the receipt up to the **Total** line. In this state, it may be possible to print tax information using the **printRecTaxID** method if this is supported by the Fiscal Printer. This state is entered from the **Fiscal Receipt** state via the **printRecVoid** method or from the **Fiscal Receipt Total** state using either the **printRecTotal**, **printRecNotPaid**, or **printRecVoid** methods. This state is exited using the **endFiscalReceipt** method at which time the Fiscal Printer returns to the **Monitor** state.
- **Fiscal Document:**
The Fiscal Printer is processing a fiscal document. The Fiscal Printer will accept the **printFiscalDocumentLine** method while in this state. This state is entered from the **Monitor** state using the **beginFiscalDocument** method. This state is exited using the **endFiscalDocument** method at which time the Fiscal Printer returns to the **Monitor** state.
- **Monitor and TrainingModeActive are true:**
The Fiscal Printer is being used for training purposes. All fiscal receipt and document commands are available. This state is entered from the **Monitor** state using the **beginTraining** method. This state is exited using the **endTraining** method at which time the Fiscal Printer returns to the **Monitor** state.
- **Fiscal Receipt and TrainingModeActive are true:**
The Fiscal Printer is being used for training purposes and a receipt is currently opened. To each line of the receipt, special text will be added in order to differentiate it from a fiscal receipt.

- **Fiscal Total** and **TrainingModeActive** are true:
The Fiscal Printer is in training mode and receipt total is being handled.
- **Fiscal ReceiptEnding** and **TrainingModeActive** are true:
The Fiscal Printer is being used for training is in the receipt ending phase.
- **NonFiscal:**
The Fiscal Printer is printing non-fiscal output on either the receipt (echoed on the journal) or the slip. In this state the Fiscal Printer will accept the **printNormal** method. The Fiscal Printer prints a message that indicates that this is non-fiscal output with all application text. This state is entered from the **Monitor** state using the **beginNonFiscal** method. This state is exited using the **endNonFiscal** method at which time the Fiscal Printer returns to the **Monitor** state.
- **Fixed:**
The Fiscal Printer is being used to print fixed, non-fiscal output to one of the Fiscal Printer's stations. In this state the Fiscal Printer will accept the **printFixedOutput** method. This state is entered from the **Monitor** state using the **beginFixedOutput** method. This state is exited using the **endFixedOutput** method at which time the Fiscal Printer returns to the **Monitor** state.
- **ItemList:**
The Fiscal Printer is currently printing a line item report. In this state the Fiscal Printer will accept the **verifyItem** method. This state is entered from the **Monitor** state using the **beginItemList** method. This state is exited using the **endItemList** method at which time the Fiscal Printer returns to the **Monitor** state.
- **Report:**
The Fiscal Printer is currently printing one of the supported types of reports. This state is entered from the **Monitor** state using one of the **printReport**, **printPeriodicTotalsReport**, **printPowerLossReport**, **printXReport** or **printZReport** methods. When the report print completes, the Fiscal Printer automatically returns to **Monitor** state.
- **FiscalSystemBlocked:**
The Fiscal Printer is no longer operational due to one of the following reasons:
 - The Fiscal Printer has been disconnected or has lost power.
 - The Fiscal Printer's fiscal memory has been exhausted.
 - The Fiscal Printer's internal data has become inconsistent.In this state the Fiscal Printer will only accept methods to print reports and retrieve data. The Fiscal Printer cannot exit this state without the assistance of an authorized technician.

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

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

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

Notice that even in this case the method returns successfully after setting **DeviceEnabled** to true. It is required that the application checks the **PowerState** property and checks for a received **StatusUpdateEvent** with the value **SUE_POWER_OFF** in the *Status* property after successfully setting the **DeviceEnabled** property.

Document Printing

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

- **Fiscal Documents:**
In order to print fiscal documents an amount value must be sent to the Fiscal Printer and recorded by it. **CapSlpFiscalDocument** will be true if the Fiscal Printer supports printing fiscal documents. If fiscal documents are supported they may be either full length (if **CapSlpFullSlip** is true) or validation (if **CapSlpValidation** is true). The actual selection is made using the **SlipSelection** property but only one totalizer is assigned to all the fiscal documents.
A fiscal document is started using the **beginFiscalDocument** method and terminated by using the **endFiscalDocument** method. A line is printed using the **printFiscalDocumentLine** method.
- **Non-Fiscal Full Length Documents:**
Full-length slip documents may be printed if **CapSlpFullSlip** is true and **SlipSelection** is set to **FPTR_SS_FULL_LENGTH**.
This document is started using the **beginNonFiscal** method and terminated by using the **endNonFiscal** method. A line is printed using the **printNormal** method.
- **Non-Fiscal Validation Documents:**
Validation documents may be printed if **CapSlpValidation** is true and **SlipSelection** is set to **FPTR_SS_VALIDATION**.
This document is started using the **beginNonFiscal** method and terminated by using the **endNonFiscal** method. A line is printed using the **printNormal** method.
- **Fixed Text Documents:**
Fixed text documents may be printed if **CapFixedOutput** is true. If fixed text documents are supported they may be either full length (if **CapSlpFullSlip** is true) or validation (if **CapSlpValidation** is true). The actual selection is made using the **SlipSelection** property.

Ordering of Fiscal Receipt Print Requests

A fiscal receipt is started using the **beginFiscalReceipt** method.

Each fiscal receipt consists of a mandatory receipt header and a mandatory receipt trailer, normally with the country specific logotype. If **CapFiscalReceiptType** is true the type of a fiscal receipt may be specified by the **FiscalReceiptType** property.

The following receipt types are defined:

- **Retail Sales Receipt:**
The daily totalizers are updated, the **printRec...** methods must be used.
- **Simplified Invoice Receipt:**
The daily totalizers are updated, a special title is printed, the **printRec...** methods can be used, except the **printRecRefund** and **printRecRefundVoid** methods.
- **Service Sales Receipt:**
The daily totalizers are updated, but a special header line is printed to identify this type of receipt. The **printRec...** methods must be used.
- **Generic Receipt:**
Free text can be printed using **printNormal** method, no totalizer is updated. A special header line is printed to identify this type of receipt.
- **Cash-In Receipt:**
This type of receipt helps to reconcile the cash amount. The cash-in amount is incremented by the amount given as an argument to the **printRecCash** method. Free text can be printed using **printNormal** method, the receipt can be cancelled.
- **Cash-Out Receipt:**
This type of receipt helps to reconcile the cash amount. The cash-in amount is decremented by the amount given as an argument to the **printRecCash** method. Free text can be printed using **printNormal** method, the receipt can be cancelled.

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

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

Once when a Retail Sales Receipt is selected and the first line item has been printed, the Fiscal Printer remains in the Fiscal Receipt state and the following fiscal print methods are available:

printRecItem
printRecItemAdjustment
printRecItemFuel
printRecItemFuelVoid
printRecPackageAdjustment
printRecPackageAdjustVoid
printRecRefund
printRecRefundVoid
printRecSubtotal
printRecSubtotalAdjustment
printRecSubtotalAdjustVoid
printRecTotal
printRecVoid
printRecVoidItem

The **printRecItem**, **printRecItemAdjustment**, **printRecItemFuel**, **printRecItemFuelVoid**, **printRecPackageAdjustment**, **printRecPackageAdjustVoid**, **printRecRefund**, **printRecRefundVoid**, **printRecSubtotal**, **printRecSubtotalAdjustment**, **printRecSubtotalAdjustVoid** and **printRecVoidItem** will leave the Fiscal Printer in the Fiscal Receipt state. The **printRecTotal** methods will change the Fiscal Printer's state to either Fiscal Receipt Total or Fiscal Receipt Ending, depending upon whether the entire receipt total has been met. The **printRecVoid** method will change the Fiscal Printer's state to Fiscal Receipt Ending.

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

printRecNotPaid
printRecTotal
printRecVoid

The **printRecNotPaid** (only available if **CapReceiptNotPaid** is true) and **printRecTotal** methods will either leave the Fiscal Printer in the Fiscal Receipt Total state or change the Fiscal Printer's state to Fiscal Receipt Ending, depending upon whether the entire receipt total has been met. The **printRecVoid** method will change the Fiscal Printer's state to Fiscal Receipt Ending.

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

printRecMessage
printRecTaxID
endFiscalReceipt

The **printRecMessage** (only available if **CapAdditionalLines** is true) and **printRecTaxID** methods will leave the Fiscal Printer in the Fiscal Receipt Ending state. The **endFiscalReceipt** will cause receipt closing and will then change the Fiscal Printer's state to Monitor.

At no time can the Fiscal Printer's total for the receipt be negative. If this occurs the Fiscal Printer will generate an **ErrorEvent**.

Fiscal Receipt Layouts *Updated in Release 1.8*

The following is an example of a typical fiscal receipt layout:

- **Header Lines:**
Header lines contain all of the information about the store, such as telephone number, address and name of the store. All of these lines are fixed and are defined before selling the first item (using the **setHeaderLine** method).
If **CapMultiContractor** property is true, two sets of header lines can be defined, assigned to the value of the **ContractorId** property. These lines may either be printed when the **beginFiscalReceipt** method is called or when the first fiscal receipt method is called.
- **Additional Header Lines:**
Header lines defined by the **AdditionalHeader** property to be printed after the fixed header lines when the **beginFiscalReceipt** method is called.
- **Transaction Lines:**
All of the lines of a fiscal transaction, such as line items, discounts and surcharges. Optionally they may be assigned to a specific contractor.
- **Total Line:**
The line containing the transaction total, tender amounts and possibly change due.
- **Message Lines:**
These are lines printed using the **printRecMessage** method.
- **Trailer Lines:**
These are fixed promotional messages stored on the Fiscal Printer (using the **setTrailerLine** method). They are automatically printed when the **endFiscalReceipt** method is called. In fact, depending upon fiscal legislation and upon the Fiscal Printer vendor, the relative position of the trailer and the fiscal logotype lines can vary.
- **Fiscal Lines:**
These are lines containing information to be inserted in the receipt due to fiscal legislations such as the fiscal logotype, date, time and serial number. They are also printed automatically when the **endFiscalReceipt** method is called.
- **Additional Trailer Lines:**
These are receipt specific information defined in the **AdditionalTrailer** property to be printed after the Fiscal Lines on the receipt before cutting it, when the **endFiscalReceipt** method is called.

Example of a Fiscal Receipt

<u>Fiscal receipt</u>	<u>Definition of the line</u>	<u>UPOS methods and properties</u>
name of the store address ZIP code and place fiscal identification of the store Good Morning	fixed header lines	beginFiscalReceipt data stored with setHeaderLine and setFiscalID AdditionalHeader property
Milk 1.000 A	transaction line	printRecItem
Special offer	pre item line	PreLine property
Beer 4.000 B	transaction line	printRecItem
Discount Beer -500 B	transaction line	printRecItemAdjustment
Bread 3.500 A	transaction line	printRecItem
Storno Bread -3.500 A	transaction line	printRecItemVoid
Apples 2.000 A	transaction line	printRecItem
 SUBTOTAL 6.500	 subtotal line	 printRecSubtotal
Lamp 12.000 C	transaction line	printRecItem
VAT category A 3.000	VAT summary	printRecTotal
VAT 7.50% 225		(... , 10000, "Check")
VAT category B 3.500		
VAT 12.00% 420		
VAT category C 12.000		
VAT 10.00% 1.200		
sum of VAT 1.845		
 TOTALE 18.500	 total line	
Check 10.000	payment line	
Cash 10.000	payment line	printRecTotal
		(... , 10000, "Cash")
Return - 1.500	change line	
Advertising messages a.s.o. THANK YOU FOR BUYING AT SABERTINI	message line trailer line trailer line	printRecMessage endFiscalReceipt data stored with setTrailerLine and at initialisation time of the fiscal printer
24/05/99 14:25 No 225	logo line	
MF B5 012345678	logo line	
Good Bye CONGRATULATION Mrs. Smith! You have won: 150 points of fidelity	additional trailer lines	AdditionalTrailer property

Totalizers and Fiscal Memory

The Fiscal Printer is able to select the fiscal relevant data and to accumulate and store them in following types of totalizers:

- **Receipt Totalizers:**
The different kind of amounts of the current receipt are accumulated in receipt totalizers.
- **Day Totalizers:**
At the end of a fiscal receipt, when calling the **endFiscalReceipt** method, the receipt totalizers are added to the day totalizers where the totals of a fiscal period (day) are summarized. The contents of the current day totalizers are printed when calling the **printXReport** method. At the end of a fiscal day or period totalizers are printed when calling the **printZReport** method.
- **Document Totalizers:**
The different kind of amounts of the current document are accumulated in document totalizers.
- **Grand Totalizers:**
Some of the totalizers are stored in the fiscal memory at the end of a fiscal period when calling the **printZReport** method. These are the grand totalizers. The application may print the contents of the fiscal memory by calling **printReport** method.

The application may fetch the different totalizers using the **getData** method or the **getTotalizer** method, whereas the type of totalizer can be specified by setting the **TotalizerType** property and the assignment to a contractor by setting the **ContractorId** property.

Counters

The Fiscal Printer is able to count some features of fiscal receipt and documents. The application may fetch the different counters using the **getData** method.

VAT Tables

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

If the Fiscal Printer supports VAT tables, **CapHasVatTable** is true. If the Fiscal Printer allows the VAT table entries to be set or modified **CapSetVatTable** is true. The maximum number of different vat rate entries in the VAT table is given by the **NumVatRates** property. VAT tables are set through a two step process. First the application uses the **setVatValue** method to set each table entry to be sent to the Fiscal Printer.

Next, the **setVatTable** method is called to send the entire VAT table to the Fiscal Printer at one time.

Receipt Duplication

In some countries, fiscal legislation can allow printing more than one copy of the same receipt. **CapDuplicateReceipt** will be true if the Fiscal Printer is capable of printing duplicate receipts. Then, setting **DuplicateReceipt** true causes the

buffering of all receipt printing commands. **DuplicateReceipt** is set false after receipt closing. In order to print the receipt again the **printDuplicateReceipt** method has to be called.

Currency amounts, percentage amounts, VAT rates, and quantity amounts

- Currency amounts (and also prices) are passed as values with the data type long. This is a 64 bit signed integer value that implicitly assumes four digits as the fractional part. For example, an actual value of 12345 represents 1.2345. So, the range supported is from
-922,337,203,685,477.5808
to
+922,337,203,685,477.5807
The fractional part used in the calculation unit of a Fiscal Printer may differ from the long data type. The number of digits in the fractional part is stored in the **AmountDecimalPlaces** property and determined by the Fiscal Printer. The application has to take care that calculations in the application use the same fractional part for amounts.
- If **CapHasVatTable** is true, VAT rates are passed using the indexes that were sent to the **setVatValue** method.
- If **CapHasVatTable** is false, VAT rates are passed as amounts with the data type *int32*. The number of digits in the fractional part is implicitly assumed to be four.
- Percentage amounts are used in methods which allow also surcharge and/or discount amounts. If the amounts are specified to be a percentage value the value is also passed in a parameter of type long.
- The percentage value has (as given by the long data type) four digits in the fractional part. It is the percentage (0.0001% to 99.9999%) multiplied by 10000.
- Quantity amounts are passed as values with the data type *int32*. The number of digits in the fractional part is stored in the **QuantityDecimalPlaces** property and determined by the Fiscal Printer.

Currency Change

If **CapSetCurrency** is true the Fiscal Printer is able to change the currency, the application may set a new currency (e.g., EURO) using the **setCurrency** method.

Device Sharing

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

- The application must claim the device before enabling it.
- The application must claim and enable the device before accessing many Fiscal 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.

Properties (UML attributes)

ActualCurrency Property

Added in Release 1.6

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

Remarks Holds a value identifying which actual currency is used by the Fiscal Printer.

This property is only valid if **CapSetCurrency** is true.

Values are:

Value	Meaning
FPTR_AC_BRC	The actual currency is Brazilian cruzeiro.
FPTR_AC_BGL	The actual currency is Bulgarian lev.
FPTR_AC_EUR	The actual currency is EURO.
FPTR_AC_GRD	The actual currency is Greek drachma.
FPTR_AC_HUF	The actual currency is Hungarian forint.
FPTR_AC_ITL	The actual currency is Italian lira.
FPTR_AC_PLZ	The actual currency is Polish zloty.
FPTR_AC_ROL	The actual currency is Romanian leu.
FPTR_AC_RUR	The actual currency is Russian rouble.
FPTR_AC_TRL	The actual currency is Turkish lira.

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 16.

See Also **setCurrency** Method, **CapSetCurrency** Property.

AdditionalHeader Property***Added in Release 1.6*****Syntax** **AdditionalHeader:** *string* { read-write, access after open-claim-enable }**Remarks** Specifies a user specific text which will be printed on the receipt after the fixed header lines when calling the **beginFiscalReceipt** method.This property is only valid if **CapAdditionalHeader** is true.

This property is initialized to an empty string 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 16.Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	The Fiscal Printer does not support printing text after the fixed header lines.

See Also **beginFiscalReceipt** Method, **CapAdditionalHeader** Property.**AdditionalTrailer Property*****Added in Release 1.6*****Syntax** **AdditionalTrailer:** *string* { read-write, access after open-claim-enable }**Remarks** Specifies a user specific text which will be printed on the receipt after the fiscal trailer lines when calling the **endFiscalReceipt** method.This property is only valid if **CapAdditionalTrailer** is true.

This property is initialized to an empty string 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 16.Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	The Fiscal Printer does not support printing text after the fiscal trailer lines.

See Also **endFiscalReceipt** Method, **CapAdditionalTrailer** Property.

AmountDecimalPlaces Property

Syntax	AmountDecimalPlaces: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	Holds the number of decimal digits that the fiscal device uses for calculations. This property is initialized when the device is enabled.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

AsyncMode Property

Syntax	AsyncMode: <i>boolean</i> { read-write, access after open }
Remarks	If true, then some print methods such as printRecItemAdjustment , printRecItem , printNormal , etc. will be performed asynchronously. If false, they will be performed synchronously. This property is initialized to false by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	“Model” on page 238 for the output model description.

CapAdditionalHeader Property

Added in Release 1.6

Syntax	CapAdditionalHeader: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer is able to print application specific text defined in the AdditionalHeader property after printing the fixed header lines. 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 16.

CapAdditionalLines Property**Updated in Release 1.8**

Syntax	CapAdditionalLines: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, then the Fiscal Printer supports the printing of application defined lines on a fiscal receipt between the total line and the end of the fiscal receipt.</p> <p>If true, then after all totals lines are printed it is possible to print application-defined strings, such as the ones used for fidelity cards.</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 16.

CapAdditionalTrailer Property**Added in Release 1.6**

Syntax	CapAdditionalTrailer: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, then the Fiscal Printer is able to print application specific text defined in the AdditionalTrailer property after printing the fiscal trailer lines.</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 16.

CapAmountAdjustment Property

Syntax	CapAmountAdjustment: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, then the Fiscal Printer handles fixed amount discounts or fixed amount surcharges on items.</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 16.

CapAmountNotPaid Property

Syntax	CapAmountNotPaid: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, then the Fiscal Printer allows the recording of not paid amounts.</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 16.

CapChangeDue Property

Added in Release 1.6

Syntax	CapChangeDue: <i>boolean</i> { read-only, access after open }
Remarks	If true, the text to be printed as the cash return description when using printRecTotal method can be defined in the ChangeDue property. 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 16.

CapCheckTotal Property

Syntax	CapCheckTotal: <i>boolean</i> { read-only, access after open }
Remarks	If true, then automatic comparison of the Fiscal Printer’s total and the application’s total can be enabled and disabled. If false, then the automatic comparison cannot be enabled and is always considered disabled. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

CapCoverSensor Property

Syntax	CapCoverSensor: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer has a “cover open” 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 16.

CapDoubleWidth Property

Syntax	CapDoubleWidth: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer can print double width 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 16.

CapDuplicateReceipt Property

Syntax	CapDuplicateReceipt: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer allows printing more than one copy of the same fiscal receipt. 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 16.

CapEmptyReceiptIsVoidable Property *Added in Release 1.6*

Syntax	CapEmptyReceiptIsVoidable: <i>boolean</i> { read-only, access after open }
Remarks	If true, then it is allowed to void an opened receipt without any items. 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 16.

CapFiscalReceiptStation Property *Added in Release 1.6*

Syntax	CapFiscalReceiptStation: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer supports printing transactions on the station defined by the FiscalReceiptStation property. 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 16.

CapFiscalReceiptType Property *Added in Release 1.6*

Syntax	CapFiscalReceiptType: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer supports printing different types of fiscal receipts defined by the FiscalReceiptType property. 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 16.

CapFixedOutput Property

Syntax	CapFixedOutput: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer supports fixed format text printing through the beginFixedOutput , printFixedOutput and endFixedOutput methods. 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 16.

CapHasVatTable Property

Syntax	CapHasVatTable: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer has a tax table. 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 16.

CapIndependentHeader Property

Syntax	CapIndependentHeader: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer supports printing the fiscal receipt header lines before the first fiscal receipt command is processed. 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 16.

CapItemList Property

Syntax	CapItemList: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer can print a report of items of a specified VAT class. 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 16.

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 16.

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 16.

CapJrnPresent Property

Syntax	CapJrnPresent: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the journal print station is present. Unlike POS printers, on Fiscal Printers the application is not able to directly access the journal. The Fiscal Printer itself prints on the journal if present. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

CapMultiContractor Property

Added in Release 1.6

Syntax	CapMultiContractor: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer supports more than one contractor assigned to the fiscal receipt and items. 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 16.

CapNonFiscalMode Property

Syntax	CapNonFiscalMode: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer allows printing in non-fiscal 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 16.

CapOnlyVoidLastItem Property

Added in Release 1.6

Syntax	CapOnlyVoidLastItem: <i>boolean</i> { read-only, access after open }
Remarks	If true, then only the last printed item can be voided. 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 16.

CapOrderAdjustmentFirst Property

Syntax	CapOrderAdjustmentFirst: <i>boolean</i> { read-only, access after open }
Remarks	If false, the application has to call printRecItem first and then call printRecItemAdjustment to give a discount or a surcharge for a single article. If true, then the application has to call printRecItemAdjustment first and then call printRecItem . 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 16.

CapPackageAdjustment Property

Added in Release 1.6

Syntax	CapPackageAdjustment: <i>boolean</i> { read-only, access after open }
Remarks	If true, an adjustment may be given to a package of booked items. 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 16.

CapPercentAdjustment Property

Syntax	CapPercentAdjustment: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer handles percentage discounts or percentage surcharges on items. 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 16.

CapPositiveAdjustment Property

Syntax	CapPositiveAdjustment: <i>boolean</i> { read-only, access after open }
Remarks	If true, then it is possible to apply surcharges via the printRecItemAdjustment method. 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 16.

CapPostPreLine Property

Added in Release 1.6

Syntax	CapPostPreLine: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer supports printing additional lines defined by the PostLine and/or the PreLine properties when calling some printRec... methods. 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 16.

CapPowerLossReport Property

Syntax	CapPowerLossReport: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer can print a power loss report using the printPowerLossReport method. 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 16.

CapPredefinedPaymentLines Property

Syntax	CapPredefinedPaymentLines: <i>boolean</i> { read-only, access after open }
Remarks	If true, the Fiscal Printer can store and print predefined payment descriptions. 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 16.

CapReceiptNotPaid Property

Syntax	CapReceiptNotPaid: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer supports using the printRecNotPaid method to specify a part of the receipt total that is not paid. 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 16.

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 16.

CapRecNearEndSensor Property

Syntax	CapRecNearEndSensor: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the receipt 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 16.

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 16.

CapRemainingFiscalMemory Property

Syntax	CapRemainingFiscalMemory: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer supports using the RemainingFiscalMemory property to show the amount of Fiscal Memory remaining. If false, the Fiscal Printer does not support reporting the Fiscal Memory status of the Fiscal Printer. 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 16.

CapReservedWord Property

Syntax	CapReservedWord: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer prints a reserved word (for example, “TOTALE”) before printing the total amount. If true, the reserved word is stored in the ReservedWord property. This reserved word may not be printed using any fiscal print method. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

CapSetCurrency Property

Added in Release 1.6

Syntax	CapSetCurrency: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer is able to change the currency to a new one by calling the setCurrency method. 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 16.

CapSetHeader Property

Syntax	CapSetHeader: <i>boolean</i> { read-only, access after open }
Remarks	If true, then it is possible to use the setHeaderLine method to initialize the contents of a particular line of the receipt header. 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 16.

CapSetPOSID Property

Syntax	CapSetPOSID: <i>boolean</i> { read-only, access after open }
Remarks	If true, then it is possible to use the setPOSID method to initialize the values of POSID and CashierID. These values are printed on each fiscal receipt. 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 16.

CapSetStoreFiscalID Property

Syntax	CapSetStoreFiscalID: <i>boolean</i> { read-only, access after open }
Remarks	If true, then it is possible to use the setStoreFiscalID method to set up the Fiscal ID number which will be printed on each fiscal receipt. 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 16.

CapSetTrailer Property

Syntax	CapSetTrailer: <i>boolean</i> { read-only, access after open }
Remarks	If true, then it is possible to use the setTrailerLine method to initialize the contents of a particular line of the receipt trailer. 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 16.

CapSetVatTable Property

Syntax	CapSetVatTable: <i>boolean</i> { read-only, access after open }
Remarks	If true, then it is possible to use the setVatValue and setVatTable methods to modify the contents of the Fiscal Printer's VAT table. Some Fiscal Printers may not allow existing VAT table entries to be modified. Only new entries may be set on these Fiscal Printers. This property is initialized by the open method.
Errors	A <code>UposException</code> may be thrown when this property is accessed. For further information, see "Errors" on page 16.

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 <code>UposException</code> may be thrown when this property is accessed. For further information, see "Errors" on page 16.

CapSlpFiscalDocument Property

Syntax	CapSlpFiscalDocument: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer allows fiscal printing to the slip station. This property is initialized by the open method.
Errors	A <code>UposException</code> may be thrown when this property is accessed. For further information, see "Errors" on page 16.

CapSlpFullSlip Property

Syntax	CapSlpFullSlip: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer supports printing full length forms on the slip station. It is possible to choose between full slip and validation documents by setting the SlipSelection property. This property is initialized by the open method.
Errors	A <code>UposException</code> may be thrown when this property is accessed. For further information, see "Errors" on page 16.

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 16.

CapSlpPresent Property

Syntax	CapSlpPresent: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer has a slip 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 16.

CapSlpValidation Property

Syntax	CapSlpValidation: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer supports printing validation information on the slip station. It is possible to choose between full slip and validation documents by setting the SlipSelection property. In some countries, when printing non fiscal validations using the slip station a limited number of lines could be printed. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

CapSubAmountAdjustment Property

Syntax	CapSubAmountAdjustment: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer handles fixed amount discounts on the subtotal. 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 16.

CapSubPercentAdjustment Property

Syntax	CapSubPercentAdjustment: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer handles percentage discounts on the subtotal. 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 16.

CapSubtotal Property

Syntax	CapSubtotal: <i>boolean</i> { read-only, access after open }
Remarks	If true, then it is possible to use the printRecSubtotal method to print the current subtotal. 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 16.

CapTotalizerType Property

Added in Release 1.6

Syntax	CapTotalizerType: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer supports reading different types of totalizers by calling the getTotalizer method. 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 16.

CapTrainingMode Property

Syntax	CapTrainingMode: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Fiscal Printer supports a training 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 16.

CapValidateJournal Property

Syntax	CapValidateJournal: <i>boolean</i> { read-only, access after open }
Remarks	If true, then it is possible to use the printNormal method to print a validation string on the journal 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 16.

CapXReport Property

Syntax	CapXReport: <i>boolean</i> { read-only, access after open }
Remarks	If true, then it is possible to use the printXReport method to print an X report. 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 16.

ChangeDue Property

Added in Release 1.6

Syntax	ChangeDue: <i>string</i> { read-write, access after open }
Remarks	This property holds the text to be printed as a description for the cash return when using the printRecTotal method. This property is only valid if CapChangeDue is true. This property is initialized to an empty string by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	Setting this property is not valid for this service (see CapChangeDue property).
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_BAD_LENGTH: The length of the string to be printed is too long.

See Also	printRecTotal Method, CapChangeDue Property.
-----------------	--

CheckTotal Property

- Syntax** **CheckTotal:** *boolean* { read-write, access after open }
- Remarks** If true, automatic comparison between the Fiscal Printer's total and the application's total is enabled. If false, automatic comparison is disabled. This property is only valid if **CapCheckTotal** is true. This property is initialized to true by the **open** method.
- Errors** A UposException may be thrown when this property is accessed. For further information, see "Errors" on page 16.

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

Value	Meaning
E_ILLEGAL	Setting this property is not valid for this Service (see CapCheckTotal).

ContractorId Property

Added in Release 1.6

- Syntax** **ContractorId:** *int32* { read-write, access after open-claim-enable }
- Remarks** The identification of the contractor to whom the receipt and/or some items of the receipt are assigned.
- It is used to define different header lines to be printed on the fiscal receipt, in order to assign any item to a specific contractor and to modify the counters and totalizers to be read using **getData** and **getTotalizer** methods.

Values are:

Value	Meaning
FPTR_CID_FIRST	First contractor is defined.
FPTR_CID_SECOND	Second contractor is defined.
FPTR_CID_SINGLE	Single contractor.

This property is initialized to FPTR_CID_SINGLE and kept current while the device is enabled, which is the functionality supported prior to Release 1.6.

- Errors** A UposException may be thrown when this property is accessed. For further information, see "Errors" on page 16.

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

Value	Meaning
E_ILLEGAL	Setting this property is not valid for this service (see CapMultiContractor property).

- See Also** **beginFiscalReceipt** Method, **getData** Method, **getTotalizer** Method, **printRec...** Methods, **CapMultiContractor** Property.

CountryCode Property**Updated in Release 1.6****Syntax** CountryCode: *int32* { read-only, access after open }**Remarks** Holds a value identifying which countries are supported by the Fiscal Printer. It can contain any of the following values logically ORed together:

Value	Meaning
FPTR_CC_BRAZIL	The Fiscal Printer supports Brazil's fiscal rules.
FPTR_CC_GREECE	The Fiscal Printer supports Greece's fiscal rules.
FPTR_CC_HUNGARY	The Fiscal Printer supports Hungary's fiscal rules.
FPTR_CC_ITALY	The Fiscal Printer supports Italy's fiscal rules.
FPTR_CC_POLAND	The Fiscal Printer supports Poland's fiscal rules.
FPTR_CC_TURKEY	The Fiscal Printer supports Turkey's fiscal rules.
FPTR_CC_RUSSIA	The Fiscal Printer supports Russia's fiscal rules.
FPTR_CC_BULGARIA	The Fiscal Printer supports Bulgaria's fiscal rules.
FPTR_CC_ROMANIA	The Fiscal Printer supports Romania's fiscal rules.

This property is initialized when the device is first enabled following the **open** method. (In releases prior to 1.5, this description stated that initialization took place by the **open** method. In Release 1.5, it was updated for consistency with other devices.)

Errors A UposException may be thrown when this property is accessed. For further information, see "Errors" on page 16.**CoverOpen Property****Syntax** CoverOpen: *boolean* { read-only, access after open-claim-enable }**Remarks** If true, then the Fiscal Printer's cover is open.

If **CapCoverSensor** is false, then the Fiscal Printer does not have a cover open sensor and 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 16.

DateType Property***Added in Release 1.6*****Syntax** **DateType: *int32* { read-write, access after open-claim-enable }****Remarks** Specifies the type of date to be requested when calling the **getDate** method.

Values are:

Value	Meaning
FPTR_DT_CONF	Date of configuration.
FPTR_DT_EOD	Date of last end of day.
FPTR_DT_RESET	Date of last reset.
FPTR_DT_RTC	Real time clock of the Fiscal Printer.
FPTR_DT_VAT	Date of last VAT change.

This property is initialized to FPTR_DT_RTC and kept current while the device is enabled, which is the functionality supported prior to Release 1.6.

Errors A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	The Fiscal Printer does not support the specified type.

See Also **getDate** Method.

DayOpened Property**Updated in Release 1.6**

Syntax	DayOpened: <i>boolean</i> { read-only, access after open-claim-enable }
Remarks	<p>If true, then the fiscal day has been started on the Fiscal Printer by a first call to the beginFiscalReceipt or beginFiscalDocument method at a fiscal period (day).</p> <p>The Fiscal Day of the Fiscal Printer can be either opened or not opened. The DayOpened property reflects whether or not the Fiscal Printer considers its Fiscal Day to be opened or not.</p> <p>Some methods may only be called while the Fiscal Day is not yet opened (DayOpened is false). Methods that can be called after the Fiscal Day is opened change from country to country. Usually all the configuration methods are to be called only before the Fiscal Day is opened.</p> <p>This property changes to false after calling printZReport.</p> <p>Depending on fiscal legislation, the following methods may be allowed only if the Fiscal Printer is in the Monitor State and has not yet begun its Fiscal Day:</p> <p style="padding-left: 40px;">setCurrency setDate setHeaderLine setPOSID setStoreFiscalID setTrailerLine setVatTable setVatValue</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 16.

DescriptionLength Property**Updated in Release 1.6**

Syntax	DescriptionLength: <i>int32</i> { read-only, access after open }
Remarks	<p>Holds the maximum number of characters that may be passed as a description parameter.</p> <p>The exact maximum number for a description parameter of a specific method can be obtained by calling getData method.</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 16.
See Also	getData Method.

DuplicateReceipt Property

Syntax	DuplicateReceipt: <i>boolean</i> { read-write, access after open }
Remarks	If true, all the printing commands inside a fiscal receipt will be buffered and they can be printed again via the printDuplicateReceipt method. This property is only valid if CapDuplicateReceipt is true. This property is initialized to false by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

ErrorLevel Property

Syntax	ErrorLevel: <i>int32</i> { read-only, access after open }										
Remarks	Holds the severity of the error condition. This property has one of the following values:										
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>FPTR_EL_NONE</td> <td>No error condition is present.</td> </tr> <tr> <td>FPTR_EL_RECOVERABLE</td> <td>A recoverable error has occurred. (Example: Out of paper.)</td> </tr> <tr> <td>FPTR_EL_FATAL</td> <td>A non-recoverable error has occurred. (Example: Internal printer failure.)</td> </tr> <tr> <td>FPTR_EL_BLOCKED</td> <td>A severe hardware failure which can be resolved only by authorized technicians. (Example: Fiscal memory failure.). This error can not be recovered.</td> </tr> </tbody> </table>	Value	Meaning	FPTR_EL_NONE	No error condition is present.	FPTR_EL_RECOVERABLE	A recoverable error has occurred. (Example: Out of paper.)	FPTR_EL_FATAL	A non-recoverable error has occurred. (Example: Internal printer failure.)	FPTR_EL_BLOCKED	A severe hardware failure which can be resolved only by authorized technicians. (Example: Fiscal memory failure.). This error can not be recovered.
Value	Meaning										
FPTR_EL_NONE	No error condition is present.										
FPTR_EL_RECOVERABLE	A recoverable error has occurred. (Example: Out of paper.)										
FPTR_EL_FATAL	A non-recoverable error has occurred. (Example: Internal printer failure.)										
FPTR_EL_BLOCKED	A severe hardware failure which can be resolved only by authorized technicians. (Example: Fiscal memory failure.). This error can not be recovered.										
	This property is set just before delivering an ErrorEvent . When the error is cleared, then the property is changed to FPTR_EL_NONE.										
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.										

ErrorOutID Property

Updated in Release 1.6

Syntax	ErrorOutID: <i>int32</i> { read-only, access after open }
Remarks	Holds the identifier of the output in the queue which caused an ErrorEvent , when using asynchronous printing. This property is initialized when the device is first enabled following the open method. (In releases prior to 1.5, this description stated that initialization took place by the open method. In Release 1.5, it was updated for consistency with other devices.) This property is set just before an ErrorEvent is delivered.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

ErrorState Property

Syntax	ErrorState: <i>int32</i> { read-only, access after open }
Remarks	Holds the current state of the Fiscal Printer when an ErrorEvent is delivered for an asynchronous output. This property is set just before an ErrorEvent is delivered.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	PrinterState Property.

ErrorStation Property

Syntax	ErrorStation: <i>int32</i> { read-only, access after open }
Remarks	Holds the station or stations that were printing when an error was detected. This property will be set to one of the following values: FPTR_S_JOURNAL, FPTR_S_RECEIPT, FPTR_S_SLIP, FPTR_S_JOURNAL_RECEIPT, FPTR_S_JOURNAL_SLIP, FPTR_S_RECEIPT_SLIP. This property is only valid if the ErrorLevel is not equal to PTR_EL_NONE. It is set just before delivering an ErrorEvent .
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

ErrorString Property

Syntax	ErrorString: <i>string</i> { read-only, access after open }
Remarks	Holds a vendor-supplied description of the current error. 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.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

FiscalReceiptStation Property**Added in Release 1.6**

- Syntax** **FiscalReceiptStation: *int32* { read-write, access after open-claim-enable }**
- Remarks** Selects the station where the transaction of the fiscal receipt started with **beginFiscalReceipt** method will be printed. Setting this property is only allowed in the Monitor State.

Values are:

Value	Meaning
FPTR_RS_RECEIPT	The following transactions will be printed on the receipt station.
FPTR_RS_SLIP	The following transactions will be printed on the slip station.

This property is only valid if **CapFiscalReceiptStation** is true.

This property is initialized to FPTR_RS_RECEIPT and kept current while the device is enabled, which is the functionality supported prior to Release 1.6.

- Errors** A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	The Fiscal Printer does not support the specified station.
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in the Monitor State.

- See Also** **beginFiscalReceipt** Method, **CapFiscalReceiptStation** Property.

FiscalReceiptType Property**Added in Release 1.6****Syntax** **FiscalReceiptType: *int32* { read-write, access after open-claim-enable }****Remarks** Selects the type of the fiscal receipt. Setting this property is only allowed in the Monitor State.

Values are:

Value	Meaning
FPTR_RT_CASH_IN	Cash-in receipt
FPTR_RT_CASH_OUT	Cash-out receipt
FPTR_RT_GENERIC	Generic receipt
FPTR_RT_SALES	Retail sales receipt
FPTR_RT_SERVICE	Service sales receipt
FPTR_RT_SIMPLE_INVOICE	Simplified invoice receipt

This property is only valid if **CapFiscalReceiptType** is true.

This property is initialized to FPTR_RT_SALES and kept current while the device is enabled, which is the functionality supported prior to Release 1.6.

Errors A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	The Fiscal Printer does not support the specified receipt type.
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in the Monitor State.

See Also **beginFiscalReceipt** Method, **CapFiscalReceiptType** Property.

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 16.

JrnEmpty Property

Syntax	JrnEmpty: <i>boolean</i> { read-only, access after open-claim-enable }
Remarks	<p>If true, the journal is out of paper. If false, journal paper is present.</p> <p>If CapJrnEmptySensor is false, then the value of this property is always false.</p> <p>This property is initialized and kept current while the device is enabled.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	JrnNearEnd Property.

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 the value of 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 16.
See Also	JrnEmpty Property.

MessageLength Property

Syntax	MessageLength: <i>int32</i> { read-only, access after open }
Remarks	Holds the maximum number of characters that may be passed as a message line in the method printRecMessage . The value may change in different modes of the Fiscal Printer. For example in the mode “Fiscal Receipt” the number of characters may be bigger than in the mode “Fiscal Receipt Total.” This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

MessageType Property**Added in Release 1.6**

Syntax	MessageType: <i>int32</i> { read-write, access after open-claim-enable }
Remarks	Selects the kind of message to be printed when using the printRecMessage method. Values are:

Value

FPTR_MT_ADVANCE
 FPTR_MT_ADVANCE_PAID
 FPTR_MT_AMOUNT_TO_BE_PAID
 FPTR_MT_AMOUNT_TO_BE_PAID_BACK
 FPTR_MT_CARD
 FPTR_MT_CARD_NUMBER
 FPTR_MT_CARD_TYPE
 FPTR_MT_CASH
 FPTR_MT_CASHIER
 FPTR_MT_CASH_REGISTER_NUMBER
 FPTR_MT_CHANGE
 FPTR_MT_CHEQUE
 FPTR_MT_CLIENT_NUMBER
 FPTR_MT_CLIENT_SIGNATURE
 FPTR_MT_COUNTER_STATE
 FPTR_MT_CREDIT_CARD
 FPTR_MT_CURRENCY
 FPTR_MT_CURRENCY_VALUE
 FPTR_MT_DEPOSIT
 FPTR_MT_DEPOSIT_RETURNED
 FPTR_MT_DOT_LINE
 FPTR_MT_DRIVER_NUMB
 FPTR_MT_EMPTY_LINE
 FPTR_MT_FREE_TEXT

FPTR_MT_FREE_TEXT_WITH_DAY_LIMIT
 FPTR_MT_GIVEN_DISCOUNT
 FPTR_MT_LOCAL_CREDIT
 FPTR_MT_MILEAGE_KM
 FPTR_MT_NOTE
 FPTR_MT_PAID
 FPTR_MT_PAY_IN
 FPTR_MT_POINT_GRANTED
 FPTR_MT_POINTS_BONUS
 FPTR_MT_POINTS_RECEIPT
 FPTR_MT_POINTS_TOTAL
 FPTR_MT_PROFITED
 FPTR_MT_RATE
 FPTR_MT_REGISTER_NUMB
 FPTR_MT_SHIFT_NUMBER
 FPTR_MT_STATE_OF_AN_ACCOUNT
 FPTR_MT_SUBSCRIPTION
 FPTR_MT_TABLE
 FPTR_MT_THANK_YOU_FOR_LOYALTY
 FPTR_MT_TRANSACTION_NUMB
 FPTR_MT_VALID_TO
 FPTR_MT_VOUCHER
 FPTR_MT_VOUCHER_PAID
 FPTR_MT_VOUCHER_VALUE
 FPTR_MT_WITH_DISCOUNT
 FPTR_MT_WITHOUT_UPLIFT

This property is initialized to `FPTR_MT_FREE_TEXT` by the **open** method, which is the functionality supported prior to Release 1.6.

Errors A `UposException` may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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

Value	Meaning
<code>E_ILLEGAL</code>	The Fiscal Printer does not support this value.

See Also `printRecMessage` Method.

NumHeaderLines Property

Syntax	NumHeaderLines: <i>int32</i> { read-only, access after open }
Remarks	Holds the maximum number of header lines that can be printed for each fiscal receipt. Header lines usually contain information such as store address, store name, store Fiscal ID. Each header line is set using the setHeaderLine method and remains set even after the Fiscal Printer is switched off. Header lines are automatically printed when a fiscal receipt is initiated using the beginFiscalReceipt method or when the first line item inside a receipt is sold. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

NumTrailerLines Property

Syntax	NumTrailerLines: <i>int32</i> { read-only, access after open }
Remarks	Holds the maximum number of trailer lines that can be printed for each fiscal receipt. Trailer lines are usually promotional messages. Each trailer line is set using the setTrailerLine method and remains set even after the Fiscal Printer is switched off. Trailer lines are automatically printed either after the last printRecTotal or when a fiscal receipt is closed using the endFiscalReceipt method. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

NumVatRates Property

Syntax	NumVatRates: <i>int32</i> { read-only, access after open }
Remarks	Holds the maximum number of vat rates that can be entered into the Fiscal Printer’s Vat table. 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 16.

PostLine Property**Added in Release 1.6**

Syntax	PostLine: <i>string</i> { read-write, access after open-claim-enable }						
Remarks	<p>An application specific text to be printed on the fiscal receipt after a line item invoked by some printRec... methods. The property can be written in the Fiscal Receipt State. The length of the text is reduced to a country specific value</p> <p>This property is only valid if CapPostPreLine is true.</p> <p>This property is initialized to an empty string and will be reset to an empty string after being used.</p>						
Errors	<p>A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>E_ILLEGAL</td> <td>The Fiscal Printer does not support printing post item lines or the text contains invalid characters.</td> </tr> <tr> <td>E_EXTENDED</td> <td><i>ErrorCodeExtended</i> = EFPTR_BAD_LENGTH: The length of the string is too long.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	The Fiscal Printer does not support printing post item lines or the text contains invalid characters.	E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_BAD_LENGTH: The length of the string is too long.
Value	Meaning						
E_ILLEGAL	The Fiscal Printer does not support printing post item lines or the text contains invalid characters.						
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_BAD_LENGTH: The length of the string is too long.						
See Also	printRecSubtotal Method, printRecTotal Method, CapPostPreLine Property.						

PredefinedPaymentLines Property

Syntax	PredefinedPaymentLines: <i>string</i> { read-only, access after open }
Remarks	<p>Holds the list of all possible words to be used as indexes of the predefined payment lines (for example, “a, b, c, d, z”). Those indexes are used in the printRecTotal method for the <i>description</i> parameter.</p> <p>If CapPredefinedPaymentLines is true, only predefined payment lines are allowed.</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 16.

PreLine Property***Added in Release 1.6***

- Syntax** **PreLine:** *string* { read-write, access after open-claim-enable }
- Remarks** An application specific text to be printed on the fiscal receipt before a line item invoked by some **printRec...** methods. The property can be written in the Fiscal Receipt State. The length of the text is reduced to a country specific value
- This property is only valid if **CapPostPreLine** is true.
- This property is initialized to an empty string and will be reset to an empty string after being used.
- Errors** A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
- Some possible values of the exception’s *ErrorCode* property are:
- | Value | Meaning |
|--------------|--|
| E_ILLEGAL | The Fiscal Printer does not support printing pre item lines or the text contains invalid characters. |
| E_EXTENDED | <i>ErrorCodeExtended</i> = EFPTR_BAD_LENGTH:
The length of the string is too long. |
- See Also** **printRecItem** Method, **printRecItemAdjustment** Method, **printRecRefund** Method, **printRecSubtotalAdjustment** Method, **CapPostPreLine** Property.

PrinterState Property**Updated in Release 1.6****Syntax** **PrinterState: *int32* { read-only, access after open }****Remarks** Holds the Fiscal Printer's current operational state. This property controls which methods are currently legal.

Values are:

Value	Meaning
FPTR_PS_MONITOR	<p>If TrainingModeActive is false: The Fiscal Printer is currently not in a specific operational mode. In this state the Fiscal Printer will accept any of the begin... methods as well as the set... methods.</p> <p>If TrainingModeActive is true: The Fiscal Printer is currently being used for training purposes. In this state the Fiscal Printer will accept any of the printRec... methods or the endTraining method.</p>
FPTR_PS_FISCAL_RECEIPT	<p>If TrainingModeActive is false: The Fiscal Printer is currently processing a fiscal receipt. In this state the Fiscal Printer will accept any of the printRec... methods.</p> <p>If TrainingModeActive is true: The Fiscal Printer is currently being used for training purposes and a fiscal receipt is currently opened.</p>
FPTR_PS_FISCAL_RECEIPT_TOTAL	<p>If TrainingModeActive is false: The Fiscal Printer has already accepted at least one payment, but the total has not been completely paid. In this state the Fiscal Printer will accept either the printRecTotal or printRecNotPaid methods.</p> <p>If TrainingModeActive is true: The Fiscal Printer is currently being used for training purposes and the Fiscal Printer has already accepted at least one payment, but the total has not been completely paid.</p>
FPTR_PS_FISCAL_RECEIPT_ENDING	<p>If TrainingModeActive is false: The Fiscal Printer has completed the receipt up to the total line. In this state the Fiscal Printer will accept either the printRecMessage or endFiscalReceipt methods.</p> <p>If TrainingModeActive is true: The Fiscal Printer is currently being used for training purposes and a fiscal receipt is going to be closed.</p>

FPTR_PS_FISCAL_DOCUMENT	The Fiscal Printer is currently processing a fiscal slip. In this state the Fiscal Printer will accept either the printFiscalDocumentLine or endFiscalDocument methods.
FPTR_PS_FIXED_OUTPUT	The Fiscal Printer is currently processing fixed text output to one or more stations. In this state the Fiscal Printer will accept either the printFixedOutput or endFixedOutput methods.
FPTR_PS_ITEM_LIST	The Fiscal Printer is currently processing an item list report. In this state the Fiscal Printer will accept either the verifyItem or endItemList methods.
FPTR_PS_NONFISCAL	The Fiscal Printer is currently processing non-fiscal output to one or more stations. In this state the Fiscal Printer will accept either the printNormal or endNonFiscal methods.
FPTR_PS_LOCKED	The Fiscal Printer has encountered a non-recoverable hardware problem. An authorized Fiscal Printer technician must be contacted to exit this state.
FPTR_PS_REPORT	The Fiscal Printer is currently processing a fiscal report. In this state the Fiscal Printer will not accept any methods until the report has completed.

There are a few methods that are accepted in any state except FPTR_PS_LOCKED. These are **beginInsertion**, **endInsertion**, **beginRemoval**, **endRemoval**, **getDate**, **getData**, **getTotalizer**, **getVatEntry**, **resetPrinter** and **clearOutput**.

This property is initialized when the device is first enabled following the **open** method. (In releases prior to 1.5, this description stated that initialization took place by the **open** method. In Release 1.5, it was updated for consistency with other devices.)

Errors A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

QuantityDecimalPlaces Property

Updated in Release 1.6

Syntax	QuantityDecimalPlaces: int32 { read-only, access after open }
Remarks	<p>Holds the number of decimal digits in the fractional part that should be assumed to be in any quantity parameter.</p> <p>This property is initialized when the device is first enabled following the open method. (In releases prior to 1.5, this description stated that initialization took place by the open method. In Release 1.5, it was updated for consistency with other devices.)</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

QuantityLength Property

Updated in Release 1.6

Syntax	QuantityLength: <i>int32</i> { read-only, access after open }
Remarks	<p>Holds the maximum number of digits that may be passed as a quantity parameter, including both the whole and fractional parts.</p> <p>This property is initialized when the device is first enabled following the open method. (In releases prior to 1.5, this description stated that initialization took place by the open method. In Release 1.5, it was updated for consistency with other devices.)</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

RecEmpty Property

Syntax	RecEmpty: <i>boolean</i> { read-only, access after open-claim-enable }
Remarks	<p>If true, the receipt is out of paper. If false, receipt paper is present.</p> <p>If CapRecEmptySensor 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 16.
See Also	RecNearEnd Property.

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 16.
See Also	RecEmpty Property.

RemainingFiscalMemory Property

Syntax	RemainingFiscalMemory: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	<p>Holds the remaining counter of Fiscal Memory.</p> <p>This property is initialized and kept current while the device is enabled and may be updated by printZReport method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	CapRemainingFiscalMemory Property.

ReservedWord Property

Syntax	ReservedWord: <i>string</i> { read-only, access after open }
Remarks	<p>Holds the string that is automatically printed with the total when the printRecTotal method is called. This word may not occur in any string that is passed into any fiscal output methods.</p> <p>This property is only valid if CapReservedWord is 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 16.

SlpEmpty Property

Syntax	SlpEmpty: <i>boolean</i> { read-only, access after open-claim-enable }
Remarks	<p>If true, a slip form is not present. If false, a slip form is present.</p> <p>If CapSlpEmptySensor is false, then this property is always false.</p> <p>This property is initialized and kept current while the device is enabled.</p> <p>Note:</p> <p><i>The “slip empty” sensor should be used primarily to determine whether a form has been inserted before printing. It can also be monitored to determine whether a form is still in place. This sensor is usually placed one or more print lines above the slip print head.</i></p> <p><i>However, the “slip near end” sensor (when present) should be used to determine when nearing the end of the slip. This sensor is usually placed one or more print lines below the slip print head.</i></p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	SlpNearEnd Property.

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. 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> <p>Note:</p> <p><i>However, the “slip near end” sensor (when present) should be used to determine when nearing the end of the slip. This sensor is usually placed one or more print lines below the slip print head.</i></p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	SlpEmpty Property.

SlipSelection Property

Syntax	SlipSelection: <i>int32</i> { read-write, access after open-claim-enable }						
Remarks	<p>Selects the kind of document to be printed on the slip station.</p> <p>This property has one of the following values:</p> <table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>FPTR_SS_FULL_LENGTH</td> <td>Print full length documents.</td> </tr> <tr> <td>FPTR_SS_VALIDATION</td> <td>Print validation documents.</td> </tr> </tbody> </table> <p>This property is initialized to FPTR_SS_FULL_LENGTH by the claim method.</p>	Value	Meaning	FPTR_SS_FULL_LENGTH	Print full length documents.	FPTR_SS_VALIDATION	Print validation documents.
Value	Meaning						
FPTR_SS_FULL_LENGTH	Print full length documents.						
FPTR_SS_VALIDATION	Print validation documents.						
Errors	<p>A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.</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 slip type was specified.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	An invalid slip type was specified.		
Value	Meaning						
E_ILLEGAL	An invalid slip type was specified.						

TotalizerType Property***Added in Release 1.6***

- Syntax** **TotalizerType: *int32* { read-write, access after open-claim-enable }**
- Remarks** Specifies the type of totalizer to be requested when calling the **getTotalizer** method.

Values are:

Value	Meaning
FPTR_TT_DOCUMENT	Document totalizer
FPTR_TT_DAY	Day totalizer
FPTR_TT_RECEIPT	Receipt totalizer
FPTR_TT_GRAND	Grand totalizer

This property is only valid if **CapTotalizerType** is true.

This property is initialized to FPTR_TT_DAY and kept current while the device is enabled, which is the functionality supported prior to Release 1.6.

- Errors** A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	The Fiscal Printer does not support defining totalizer types or an invalid type was specified.

- See Also** **getTotalizer** Method, **CapTotalizerType** Property.

TrainingModeActive Property

Syntax **TrainingModeActive:** *boolean* { **read-only, access after open-claim-enable** }

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

This property has one of the following values:

Value	Meaning
true	The Fiscal Printer is currently in training mode. That means no data are written into the EPROM of the Fiscal Printer.
false	The Fiscal Printer is currently in normal mode. All printed receipts will also update the fiscal memory.

Errors A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

Methods (UML operations)

beginFiscalDocument Method

Updated in Release 1.6

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

Parameter	Description
<i>documentAmount</i>	Amount of document to be stored by the Fiscal Printer.

Remarks Initiates fiscal printing to the slip station.

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

If this is the first call to the **beginFiscalDocument** method, the Fiscal Day will be started and the **DayOpened** property will be set to true.

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

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

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

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	The slip station does not exist (see the CapSlpPresent property) or the printer does not support fiscal output to the slip station (see the CapSlpFiscalDocument property).
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The printer’s current state does not allow this state transition. <i>ErrorCodeExtended</i> = EFPTR_SLP_EMPTY: There is no paper in the slip station. <i>ErrorCodeExtended</i> = EFPTR_BAD_ITEM_AMOUNT: The <i>documentAmount</i> parameter is invalid. <i>ErrorCodeExtended</i> = EFPTR_MISSING_SET_CURRENCY: The new receipt cannot be opened, the Fiscal Printer is expecting the current currency to be changed by calling setCurrency method.

See Also **CapSlpFiscalDocument** Property, **CapSlpPresent** Property, **AmountDecimalPlaces** Property, **DayOpened** Property, **PrinterState** Property, **beginInsertion** Method, **endFiscalDocument** Method, **endInsertion** Method, **printFiscalDocumentLine** Method, **printZReport** Method.

beginFiscalReceipt Method**Updated in Release 1.6**

Syntax	beginFiscalReceipt (printHeader: <i>boolean</i>): void { raises-exception, use after open-claim-enable }						
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Parameter</th> <th style="text-align: left;">Description</th> </tr> </thead> <tbody> <tr> <td><i>printHeader</i></td> <td>Indicates if the header lines are to be printed at this time.</td> </tr> </tbody> </table>	Parameter	Description	<i>printHeader</i>	Indicates if the header lines are to be printed at this time.		
Parameter	Description						
<i>printHeader</i>	Indicates if the header lines are to be printed at this time.						
Remarks	<p>Initiates fiscal printing to the receipt station.</p> <p>If CapFiscalReceiptStation is true the FiscalReceiptStation property defines the station where the receipt will be printed. If CapFiscalReceiptStation is false the receipt will be printed on the receipt station. If CapFiscalReceiptType is true the receipt type must be defined in FiscalReceiptType and a header line according to the specified receipt type will be printed.</p> <p>If this is the first call to the beginFiscalReceipt method, the Fiscal Day will be started and the DayOpened property will be set to true.</p> <p>If <i>printHeader</i> and CapIndependentHeader are both true all defined header lines will be printed before control is returned. Otherwise, header lines will be printed when the first item is sold in the case they are not printed at the end of the preceding receipt. If CapAdditionalHeader is true, application specific header lines defined by the AdditionalHeader property will be printed after the fixed header lines.</p> <p>If CapMultiContractor is true, the current receipt is assigned to the contractor specified by the ContractorId property.</p> <p>If this method is successful, the PrinterState property will be changed to FPTR_PS_FISCAL_RECEIPT.</p>						
Errors	<p>A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Value</th> <th style="text-align: left;">Meaning</th> </tr> </thead> <tbody> <tr> <td>E_ILLEGAL</td> <td>An invalid receipt type was specified.</td> </tr> <tr> <td>E_EXTENDED</td> <td><i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer’s current state does not allow this state transition. <i>ErrorCodeExtended</i> = EFPTR_MISSING_SET_CURRENCY: The new receipt cannot be opened, the Fiscal Printer is expecting the current currency to be changed by calling setCurrency method.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	An invalid receipt type was specified.	E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer’s current state does not allow this state transition. <i>ErrorCodeExtended</i> = EFPTR_MISSING_SET_CURRENCY: The new receipt cannot be opened, the Fiscal Printer is expecting the current currency to be changed by calling setCurrency method.
Value	Meaning						
E_ILLEGAL	An invalid receipt type was specified.						
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer’s current state does not allow this state transition. <i>ErrorCodeExtended</i> = EFPTR_MISSING_SET_CURRENCY: The new receipt cannot be opened, the Fiscal Printer is expecting the current currency to be changed by calling setCurrency method.						
See Also	CapAdditionalHeader Property, CapFiscalReceiptStation Property, CapFiscalReceiptType Property, CapIndependentHeader Property, CapMultiContractor Property, AdditionalHeader Property, ContractorId Property, DayOpened Property, FiscalReceiptStation Property, FiscalReceiptType Property, PrinterState Property, endFiscalReceipt Method, printRec... Methods.						

beginFixedOutput Method

Syntax **beginFixedOutput (station: *int32*, documentType: *int32*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>station</i>	The Fiscal Printer station to be used. May be either FPTR_S_RECEIPT or FPTR_S_SLIP.
<i>documentType</i>	Identifier of a document stored in the Fiscal Printer.

Remarks Initiates non-fiscal fixed text printing on a Fiscal Printer station. This method is only supported if **CapFixedOutput** is true.

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

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

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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"> • Station does not exist (see the CapSlpPresent property). • Fiscal Printer does not support fixed output (see the CapFixedOutput property). • <i>station</i> parameter is invalid. • <i>documentType</i> is invalid.
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer’s current state does not allow this state transition.
	<i>ErrorCodeExtended</i> = EFPTR_SLP_EMPTY: There is no paper in the slip station.

See Also **CapFixedOutput** Property, **CapSlpPresent** Property, **PrinterState** Property, **beginInsertion** Method, **endFixedOutput** Method, **endInsertion** Method, **printFixedOutput** Method.

beginInsertion Method

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

Parameter	Description
<i>timeout</i>	The <i>timeout</i> parameter gives the number of milliseconds before failing the method.

If zero, the method tries to begin insertion mode, then returns the appropriate status immediately. If FOREVER (-1), the method tries to begin insertion mode, then waits as long as needed until either the form is inserted or an error occurs.

Remarks Initiates slip processing.

When called, the slip station is made ready to receive a form by opening the form's handling "jaws" or activating a form insertion mode. This method is paired with the **endInsertion** method for controlling form insertion.

If the Fiscal Printer device cannot be placed into insertion mode, a UposException is thrown. Otherwise, the device continues to monitor form insertion until either:

- The form is successfully inserted.
- The form is not inserted before *timeout* milliseconds have elapsed, or an error is reported by the Fiscal Printer device. In this case, a UposException is thrown with an *ErrorCode* of E_TIMEOUT or another value. The Fiscal 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 16.

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

Value	Meaning
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 **CapSlpPresent** Property, **endInsertion** Method, **beginRemoval** Method, **endRemoval** Method.

beginItemList Method

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

Parameter	Description
-----------	-------------

<i>vatID</i>	Vat identifier for reporting.
--------------	-------------------------------

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

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

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

After this method, only **verifyItem** and **endItemList** methods may be called.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	The Fiscal Printer does not support an item list report (see the CapItemList property) or the Fiscal Printer does not support VAT tables (see the CapHasVatTable property).
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer’s current state does not allow this state transition. <i>ErrorCodeExtended</i> = EFPTR_BAD_VAT: The <i>vatID</i> parameter is invalid.

See Also **CapHasVatTable** Property, **CapItemList** Property, **PrinterState** Property, **endItemList** Method, **verifyItem** Method.

beginNonFiscal Method

Syntax	beginNonFiscal (): void { raises-exception, use after open-claim-enable }						
Remarks	<p>Initiates non-fiscal operations on the Fiscal Printer.</p> <p>This method is only supported if CapNonFiscalMode is true. Output in this mode is accomplished using the printNormal method. This method can be successfully called only if the current value of the PrinterState property is FPTR_PS_MONITOR. If this method is successful, the PrinterState property will be changed to FPTR_PS_NONFISCAL. In order to stop non fiscal modality endNonFiscal method should be called.</p>						
Errors	<p>A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.</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>The Fiscal Printer does not support non-fiscal output (see the CapNonFiscalMode property).</td> </tr> <tr> <td>E_EXTENDED</td> <td><i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer’s current state does not allow this state transition.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	The Fiscal Printer does not support non-fiscal output (see the CapNonFiscalMode property).	E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer’s current state does not allow this state transition.
Value	Meaning						
E_ILLEGAL	The Fiscal Printer does not support non-fiscal output (see the CapNonFiscalMode property).						
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer’s current state does not allow this state transition.						
See Also	CapNonFiscalMode Property, PrinterState Property, endNonFiscal Method, printNormal Method.						

beginRemoval Method

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

Parameter	Description
-----------	-------------

<i>timeout</i>	The <i>timeout</i> parameter gives the number of milliseconds before failing the method.
----------------	--

If zero, the method tries to begin removal mode, then returns the appropriate status immediately. If FOREVER (-1), the method tries to begin removal mode, then waits as long as needed until either the form is removed or an error occurs.

Remarks Initiates form removal processing.

When called, the Fiscal 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 Fiscal Printer device cannot be placed into removal or ejection mode, a *UposException* is thrown. Otherwise, the device continues to monitor form removal until either:

- The form is successfully removed.
- The form is not removed before *timeout* milliseconds have elapsed, or an error is reported by the Fiscal Printer device. In this case, a *UposException* is thrown with an *ErrorCode* of *E_TIMEOUT* or another value. The Fiscal 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 16.

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

Value	Meaning
<i>E_ILLEGAL</i>	The Fiscal Printer does not have a slip station (see the CapSlpPresent property) or an invalid <i>timeout</i> parameter was specified.
<i>E_TIMEOUT</i>	The specified time has elapsed without the form being properly removed.

See Also **CapSlpPresent** Property, **beginInsertion** Method, **endInsertion** Method, **endRemoval** Method.

beginTraining Method

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

Remarks Initiates training operations.

This method is only supported if **CapTrainingMode** is true. Output in this mode is accomplished using the **printRec...** methods in order to print a receipt or other methods to print reports. This method can be successfully called only if the current value of the **PrinterState** property is FPTR_PS_MONITOR. If this method is successful, the **TrainingModeActive** property will be changed to true.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	The Fiscal Printer does not support training mode (see the CapTrainingMode property).
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer’s current state does not allow this state transition.

See Also **CapTrainingMode** Property, **PrinterState** Property, **TrainingModeActive** Property, **endTraining** Method, **printRec...** Methods.

clearError Method

- Syntax** **clearError ():**
 void { raises-exception, use after open-claim-enable }
- Remarks** Clears all Fiscal Printer error conditions.
 This method is always performed synchronously.
- Errors** A UposException may be thrown when this method is invoked. For further
 information, see “Errors” on page 16.

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

Value	Meaning
E_FAILURE	Error recovery failed.

endFiscalDocument Method

- Syntax** **endFiscalDocument ():**
 void { raises-exception, use after open-claim-enable }
- Remarks** Terminates fiscal printing to the slip station.

 This method is only supported if **CapSlpFiscalDocument** is true.
 If this method is successful, the **PrinterState** property will be changed to
 FPTR_PS_MONITOR.
- Errors** A UposException may be thrown when this method is invoked. For further
 information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	The Fiscal Printer does not support fiscal output to the slip station (see the CapSlpFiscalDocument property).
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in the Fiscal Document state.

- See Also** **CapSlpFiscalDocument** Property, **PrinterState** property,
 beginFiscalDocument Method, **printFiscalDocumentLine** Method.

endFiscalReceipt Method**Updated in Release 1.6**

Syntax	endFiscalReceipt (printHeader: <i>boolean</i>): void { raises-exception, use after open-claim-enable }				
	<table> <thead> <tr> <th style="text-align: left;">Parameter</th> <th style="text-align: left;">Description</th> </tr> </thead> <tbody> <tr> <td><i>printHeader</i></td> <td>Indicates if the header lines of the following receipt are to be printed at this time.</td> </tr> </tbody> </table>	Parameter	Description	<i>printHeader</i>	Indicates if the header lines of the following receipt are to be printed at this time.
Parameter	Description				
<i>printHeader</i>	Indicates if the header lines of the following receipt are to be printed at this time.				
Remarks	<p>Terminates fiscal printing to the receipt station.</p> <p>If <i>printHeader</i> is false, this method will close the current fiscal receipt, print the trailer lines, if they were not already printed after the total lines, and cut it.</p> <p>If <i>printHeader</i> is true additionally the header of the next receipt will be printed before cutting the receipt, otherwise the header will be printed when beginning the next receipt.</p> <p>All functions carried out by this method will be completed before this call returns.</p> <p>If CapAdditionalTrailer is true application specific trailer lines defined by the AdditionalTrailer property will be printed after the fiscal trailer lines.</p> <p>If this method is successful, the PrinterState property will be changed to FPTR_PS_MONITOR.</p>				
Errors	<p>A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table> <thead> <tr> <th style="text-align: left;">Value</th> <th style="text-align: left;">Meaning</th> </tr> </thead> <tbody> <tr> <td>E_EXTENDED</td> <td><i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in the Fiscal Receipt Ending state.</td> </tr> </tbody> </table>	Value	Meaning	E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in the Fiscal Receipt Ending state.
Value	Meaning				
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in the Fiscal Receipt Ending state.				
See Also	beginFiscalReceipt Method, printRec... Methods, CapAdditionalTrailer Property, AdditionalTrailer Property.				

endFixedOutput Method

Syntax	endFixedOutput (): void { raises-exception, use after open-claim-enable }						
Remarks	Terminates non-fiscal fixed text printing on a Fiscal Printer station. This method is only supported if CapFixedOutput is true. If this method is successful, the PrinterState property will be changed to FPTR_PS_MONITOR.						
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16. Some possible values of the exception’s <i>ErrorCode</i> property are:						
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>E_ILLEGAL</td> <td>The Fiscal Printer does not support fixed output (see the CapFixedOutput property).</td> </tr> <tr> <td>E_EXTENDED</td> <td><i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in the Fixed Output state.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	The Fiscal Printer does not support fixed output (see the CapFixedOutput property).	E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in the Fixed Output state.
Value	Meaning						
E_ILLEGAL	The Fiscal Printer does not support fixed output (see the CapFixedOutput property).						
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in the Fixed Output state.						
See Also	beginFixedOutput Method, printFixedOutput Method.						

endInsertion Method

Syntax	endInsertion (): void { raises-exception, use after open-claim-enable }						
Remarks	Ends form insertion processing. When called, the Fiscal Printer is taken out of form insertion mode. If the slip device has forms “jaws,” they are closed by this method. If no form is present, a UposException is thrown with its <i>ErrorCodeExtended</i> property set to EFPTR_SLP_EMPTY. This method is paired with the beginInsertion method for controlling form insertion. The application may choose to call this method immediately after a successful beginInsertion if it wants to use the Fiscal Printer sensors to determine when a form is positioned within the slip printer. Alternatively, the application may prompt the user and wait for a key press before calling this method.						
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16. Some possible values of the exception’s <i>ErrorCode</i> property are:						
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>E_ILLEGAL</td> <td>The Fiscal Printer is not in slip insertion mode.</td> </tr> <tr> <td>E_EXTENDED</td> <td><i>ErrorCodeExtended</i> = EFPTR_COVER_OPEN: The device was taken out of insertion mode while the Fiscal Printer cover was open. <i>ErrorCodeExtended</i> = EFPTR_SLP_EMPTY: The device was taken out of insertion mode without a form being inserted.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	The Fiscal Printer is not in slip insertion mode.	E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_COVER_OPEN: The device was taken out of insertion mode while the Fiscal Printer cover was open. <i>ErrorCodeExtended</i> = EFPTR_SLP_EMPTY: The device was taken out of insertion mode without a form being inserted.
Value	Meaning						
E_ILLEGAL	The Fiscal Printer is not in slip insertion mode.						
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_COVER_OPEN: The device was taken out of insertion mode while the Fiscal Printer cover was open. <i>ErrorCodeExtended</i> = EFPTR_SLP_EMPTY: The device was taken out of insertion mode without a form being inserted.						
See Also	beginInsertion Method, beginRemoval Method, endRemoval Method.						

endItemList Method

- Syntax** **endItemList ():**
 void { raises-exception, use after open-claim-enable }
- Remarks** Terminates a validation report of items belonging to a particular VAT class. This method is only supported if **CapItemList** is true and **CapHasVatTable** is true. This method is paired with the **beginItemList** method. This method can be successfully called only if current value of **PrinterState** property is equal to **FPTR_PS_ITEM_LIST**. If this method is successful, the **PrinterState** property will be changed to **FPTR_PS_MONITOR**.
- Errors** A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	The Fiscal Printer does not support fixed output (see the CapItemList property) or the Fiscal Printer does not support VAT tables (see the CapHasVatTable property).
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer’s current state does not allow this state transition.

- See Also** **beginItemList** Method, **verifyItem** Method.

endNonFiscal Method

- Syntax** **endNonFiscal ():**
 void { raises-exception, use after open-claim-enable }
- Remarks** Terminates non-fiscal operations on one Fiscal Printer station. This method is only supported if **CapNonFiscalMode** is true. If this method is successful, the **PrinterState** property will be changed to **FPTR_PS_MONITOR**.
- Errors** A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	The Fiscal Printer does not support non-fiscal output (see the CapNonFiscalMode property).
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in the Non-Fiscal state.

- See Also** **beginNonFiscal** Method, **printNormal** Method.

endRemoval Method

Syntax	endRemoval (): void { raises-exception, use after open-claim-enable }						
Remarks	<p>Ends form removal processing.</p> <p>When called, the Fiscal Printer is taken out of form removal or ejection mode. If a form is present, a UposException is thrown with the <i>ErrorCodeExtended</i> property set to EFPTR_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 Fiscal 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 16.</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>The Fiscal Printer is not in slip removal mode.</td> </tr> <tr> <td>E_EXTENDED</td> <td><i>ErrorCodeExtended</i> = EFPTR_SLP_FORM: The device was taken out of removal mode while a form was still present.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	The Fiscal Printer is not in slip removal mode.	E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_SLP_FORM: The device was taken out of removal mode while a form was still present.
Value	Meaning						
E_ILLEGAL	The Fiscal Printer is not in slip removal mode.						
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_SLP_FORM: The device was taken out of removal mode while a form was still present.						
See Also	beginInsertion Method, endInsertion Method, beginRemoval Method.						

endTraining Method

Syntax	endTraining (): void { raises-exception, use after open-claim-enable }						
Remarks	<p>Terminates training operations on either the receipt or the slip station.</p> <p>This method is only supported if CapTrainingMode is true. If this method is successful, the TrainingModeActive property will be changed to false.</p>						
Errors	<p>A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.</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>The Fiscal Printer does not support training mode (see the CapTrainingMode property).</td> </tr> <tr> <td>E_EXTENDED</td> <td><i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in the Training state.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	The Fiscal Printer does not support training mode (see the CapTrainingMode property).	E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in the Training state.
Value	Meaning						
E_ILLEGAL	The Fiscal Printer does not support training mode (see the CapTrainingMode property).						
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in the Training state.						
See Also	CapTrainingMode property, beginTraining Method, printRec... Methods.						

getData Method**Updated in Release 1.6**

Syntax **getData (dataItem: *int32*, inout optArgs: *int32*, inout data: *string*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>dataItem</i>	The specific data item to retrieve.
<i>optArgs</i>	For some <i>dataItem</i> this additional argument is needed. Consult the Service vendor's documentation for further use of this argument.
<i>data</i>	Character string to hold the data retrieved.

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

Value	Meaning
--------------	----------------

Identification data

FPTR_GD_FIRMWARE	Get the Fiscal Printer's firmware release number.
FPTR_GD_PRINTER_ID	Get the Fiscal Printer's fiscal ID.

Totals

FPTR_GD_CURRENT_TOTAL	Get the current receipt total.
FPTR_GD_DAILY_TOTAL	Get the daily total.
FPTR_GD_GRAND_TOTAL	Get the Fiscal Printer's grand total.
FPTR_GD_MID_VOID	Get the total number of voided receipts.
FPTR_GD_NOT_PAID	Get the current total of not paid receipts.
FPTR_GD_RECEIPT_NUMBER	Get the number of fiscal receipts printed.
FPTR_GD_REFUND	Get the current total of refunds.
FPTR_GD_REFUND_VOID	Get the current total of voided refunds.

Fiscal memory counts

FPTR_GD_NUMB_CONFIG_BLOCK	Get the grand number of configuration blocks.
FPTR_GD_NUMB_CURRENCY_BLOCK	Get the grand number of currency blocks.
FPTR_GD_NUMB_HDR_BLOCK	Get the grand number of header blocks.
FPTR_GD_NUMB_RESET_BLOCK	Get the grand number of reset blocks.
FPTR_GD_NUMB_VAT_BLOCK	Get the grand number of VAT blocks.

Counter

FPTR_GD_FISCAL_DOC	Get the number of daily fiscal documents.
FPTR_GD_FISCAL_DOC_VOID	Get the number of daily voided fiscal documents.
FPTR_GD_FISCAL_REC	Get the number of daily fiscal sales receipts.
FPTR_GD_FISCAL_REC_VOID	Get the number of daily voided fiscal sales receipts.
FPTR_GD_NONFISCAL_DOC	Get the number of daily non fiscal documents.
FPTR_GD_NONFISCAL_DOC_VOID	Get the number of daily voided non fiscal documents.
FPTR_GD_NONFISCAL_REC	Get the number of daily non fiscal receipts.
FPTR_GD_RESTART	Get the Fiscal Printer's restart count
FPTR_GD_SIMP_INVOICE	Get the number of daily simplified invoices.
FPTR_GD_Z_REPORT	Get the Z report number.

Fixed fiscal printer text

FPTR_GD_TENDER	Get the payment description used in the printRecTotal method, defined by the given identifier in the <i>optArgs</i> argument. Valid only, if the CapPredefinedPaymentLines property is true.
----------------	--

Linecounter

FPTR_GD_LINECOUNT	Get the number of printed lines, defined by the given identifier in the <i>optArgs</i> argument. If the CapMultiContractor property is true, line counters depend on the contractor defined by the ContractorId property.
-------------------	---

Description length

FPTR_GD_DESCRIPTION_LENGTH	Get the maximum number of characters that may be passed as a description parameter for a specific method, defined by the given identifier in the <i>optArgs</i> argument.
----------------------------	---

If *dataItem* is FPTR_GD_TENDER the *optArgs* parameter has to be set to one of the following values:

Value	Meaning
FPTR_PDL_CASH	Cash.
FPTR_PDL_CHEQUE	Cheque.
FPTR_PDL_CHITTY	Chitty.
FPTR_PDL_COUPON	Coupon.
FPTR_PDL_CURRENCY	Currency.
FPTR_PDL_DRIVEN_OFF	
FPTR_PDL_EFT_IMPRINTER	Printer EFT.
FPTR_PDL_EFT_TERMINAL	Terminal EFT.
FPTR_PDL_TERMINAL_IMPRINTER	
FPTR_PDL_FREE_GIFT	Gift.
FPTR_PDL_GIRO	Giro.
FPTR_PDL_HOME	Home.
FPTR_PDL_IMPRINTER_WITH_ISSUER	
FPTR_PDL_LOCAL_ACCOUNT	Local account.
FPTR_PDL_LOCAL_ACCOUNT_CARD	Local card account.
FPTR_PDL_PAY_CARD	Pay card.
FPTR_PDL_PAY_CARD_MANUAL	Manual pay card.
FPTR_PDL_PREPAY	Prepay.
FPTR_PDL_PUMP_TEST	Pump test.
FPTR_PDL_SHORT_CREDIT	Credit.
FPTR_PDL_STAFF	Staff.
FPTR_PDL_VOUCHER	Voucher.

If *dataItem* is FPTR_GD_LINECOUNT the *optArgs* parameter has to be set to one of the following values:

Value	Meaning
FPTR_LC_ITEM	Number of item lines.
FPTR_LC_ITEM_VOID	Number of voided item lines.
FPTR_LC_DISCOUNT	Number of discount lines.
FPTR_LC_DISCOUNT_VOID	Number of voided discount lines.
FPTR_LC_SURCHARGE	Number of surcharge lines.
FPTR_LC_SURCHARGE_VOID	Number of voided surcharge lines.
FPTR_LC_REFUND	Number of refund lines.
FPTR_LC_REFUND_VOID	Number of voided refund lines.
FPTR_LC_SUBTOTAL_DISCOUNT	Number of subtotal discount lines.
FPTR_LC_SUBTOTAL_DISCOUNT_VOID	Number of voided subtotal discount lines.
FPTR_LC_SUBTOTAL_SURCHARGE	Number of subtotal surcharge lines.
FPTR_LC_SUBTOTAL_SURCHARGE_VOID	Number of voided subtotal surcharge lines.
FPTR_LC_COMMENT	Number of comment lines.
FPTR_LC_SUBTOTAL	Number of subtotal lines.
FPTR_LC_TOTAL	Number of total lines.

If *dataItem* is FPTR_GD_DESCRIPTION_LENGTH the *optArgs* parameter has to be set to one of the following values:

Value	Meaning
FPTR_DL_ITEM	printRecItem method.
FPTR_DL_ITEM_ADJUSTMENT	printRecItemAdjustment method.
FPTR_DL_ITEM_FUEL	printRecItemFuel method.
FPTR_DL_ITEM_FUEL_VOID	printRecItemFuelVoid method.
FPTR_DL_NOT_PAID	printRecNotPaid method.
FPTR_DL_PACKAGE_ADJUSTMENT	printRecPackageAdjustment method.
FPTR_DL_REFUND	printRecRefund method.
FPTR_DL_REFUND_VOID	printRecRefundVoid method.
FPTR_DL_SUBTOTAL_ADJUSTMENT	printRecSubtotalAdjustment method.
FPTR_DL_TOTAL	printRecTotal method.
FPTR_DL_VOID	printRecVoid method.
FPTR_DL_VOID_ITEM	printRecVoidItem method.

Remarks Retrieves data and counters from the printer’s fiscal module.

If **CapMultiContractor** is true, line counters depend on the contractor defined by the **ContractorId** property.

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

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_BUSY	Cannot perform while output is in progress. (Only applies if AsyncMode is false.)
E_ILLEGAL	The <i>dataItem</i> , <i>optArgs</i> or ContractorId specified is invalid.

See Also **printRecTotal** Method, **CapPredefinedPaymentLines** Property, **ContractorId** Property, **PredefinedPaymentLines** Property.

getDate Method**Updated in Release 1.6**

Syntax **getDate (inout date: *string*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>date</i>	Date and time returned as a string.

Remarks Gets the Fiscal Printer's date and time specified by the **DateType** property.

The date and time are returned as a string in the format "ddmmyyyhhmm":

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

The fiscal controller may not support hours and minutes depending on the date type. In such cases the corresponding fields in the returned string are filled with "0".

Errors A UposException may be thrown when this method is invoked. For further information, see "Errors" on page 16.

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

Value	Meaning
E_ILLEGAL	Retrieval of the date and time is not valid at this time.

See Also **DateType** Property.

getTotalizer Method**Updated in Release 1.6**

Syntax **getTotalizer (vatID: *int32*, optArgs: *int32*, inout data: *string*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>vatID</i>	VAT identifier of the required totalizer.
<i>optArgs</i>	Specifies the required totalizer.
<i>data</i>	Totalizer returned as a string.

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

Value	Meaning
FPTR_GT_GROSS	Gross totalizer specified by the TotalizerType and ContractorId properties.
FPTR_GT_NET	Net totalizer specified by the TotalizerType and ContractorId properties.
FPTR_GT_DISCOUNT	Discount totalizer specified by the TotalizerType and ContractorId properties.
FPTR_GT_DISCOUNT_VOID	Voided discount totalizer specified by the TotalizerType and ContractorId properties.
FPTR_GT_ITEM	Item totalizer specified by the TotalizerType and ContractorId properties.
FPTR_GT_ITEM_VOID	Voided item totalizer specified by the TotalizerType and ContractorId properties.
FPTR_GT_NOT_PAID	Not paid totalizer specified by the TotalizerType and ContractorId properties.
FPTR_GT_REFUND	Refund totalizer specified by the TotalizerType and ContractorId properties.
FPTR_GT_REFUND_VOID	Voided refund totalizer specified by the TotalizerType and ContractorId properties.
FPTR_GT_SUBTOTAL_DISCOUNT	Subtotal discount totalizer specified by the TotalizerType and ContractorId properties.
FPTR_GT_SUBTOTAL_DISCOUNT_VOID	Voided discount totalizer specified by the TotalizerType and ContractorId properties.
FPTR_GT_SUBTOTAL_SURCHARGES	Subtotal surcharges totalizer specified by the TotalizerType and ContractorId properties.

FPTR_GT_SUBTOTAL_SURCHARGES_VOID	Voided surcharges totalizer specified by the TotalizerType and ContractorId properties.
FPTR_GT_SURCHARGE	Surcharge totalizer specified by the TotalizerType and ContractorId properties.
FPTR_GT_SURCHARGE_VOID	Voided surcharge totalizer specified by the TotalizerType and ContractorId properties.
FPTR_GT_VAT	VAT totalizer specified by the TotalizerType and ContractorId properties.
FPTR_GT_VAT_CATEGORY	VAT totalizer per VAT category specified by the TotalizerType and ContractorId properties associated to the given <i>vatID</i> .

Remarks Gets the totalizer specified by the *optArgs* argument. Some of the totalizers such as item or VAT totalizers may be associated with the given *vatID*.

If **CapTotalizerType** is true the type of totalizer (grand, day, receipt specific) depends on the **TotalizerType** property.

If **CapMultiContractor** is true the type depends on the **ContractorId** property.

If **CapSetVatTable** is false, then only one totalizer is present.

Errors A *UposException* may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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"> • The <i>vatID</i> parameter is invalid, or • The ContractorId property is invalid, or • The specified totalizer is not available.

See Also **CapTotalizerType** Property, **TotalizerType** Property, **CapMultiContractor** Property, **ContractorId** Property.

getVatEntry Method

Syntax **getVatEntry (vatID: *int32*, optArgs: *int32*, inout vatRate: *int32*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>vatID</i>	VAT identifier of the required rate.
<i>optArgs</i>	For some countries, this additional argument may be needed. Consult the Fiscal Printer Service vendor's documentation for details.
<i>vatRate</i>	The rate associated with the VAT identifier.

Remarks Gets the rate associated with a given VAT identifier.

This method is only supported if **CapSetVatTable** is true.

Errors A UposException may be thrown when this method is invoked. For further information, see "Errors" on page 16.

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

Value	Meaning
E_ILLEGAL	The <i>vatID</i> parameter is invalid, or CapSetVatTable is false.

See Also **CapSetVatTable** Property.

printDuplicateReceipt Method

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

Remarks Prints a duplicate of a buffered transaction.

This method is only supported if **CapDuplicateReceipt** is true. This method will succeed if both the **CapDuplicateReceipt** and **DuplicateReceipt** properties are true.

This method resets the **DuplicateReceipt** property to false.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_BUSY	Cannot perform while output is in progress. (Only applies if AsyncMode is false.)
E_ILLEGAL	The Fiscal Printer does not support duplicate receipts (see the CapDuplicateReceipt property) or there is no buffered transaction to print (see DuplicateReceipt property).
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in the Monitor state. <i>ErrorCodeExtended</i> = EFPTR_JRN_EMPTY: The journal station is out of paper. <i>ErrorCodeExtended</i> = EFPTR_REC_EMPTY: The receipt station is out of paper.

See Also **CapDuplicateReceipt** Property, **DuplicateReceipt** Property.

printFiscalDocumentLine Method

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

Parameter	Description
<i>documentLine</i>	String to be printed on the fiscal slip.

Remarks Prints a line of fiscal text to the slip station.

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

Errors A UposException may be thrown when this method is invoked. For further
 information, see “Errors” on page 16.

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

Value	Meaning
E_BUSY	Cannot perform while output is in progress. (Only applies if AsyncMode is false.)
E_ILLEGAL	The Fiscal Printer does not support fiscal documents (see the CapSlpFiscalDocument property).
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in the Fiscal Document state. <i>ErrorCodeExtended</i> = EFPTR_COVER_OPEN: The Fiscal Printer cover is open. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted. (Only applies if AsyncMode is false.)

See Also **beginFiscalDocument** Method, **endFiscalDocument** Method.

printFixedOutput Method

Syntax **printFixedOutput (documentType: *int32*, lineNumber: *int32*, data: *string*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>documentType</i>	Identifier of a document stored in the Fiscal Printer
<i>lineNumber</i>	Number of the line in the document to print.
<i>data</i>	String parameter for placement in printed line.

Remarks Prints a line of a fixed document to the print station specified in the **beginFixedOutput** method. Each call prints a single line from a document by merging the stored text with the parameter *data*. Within a document lines must be printed sequentially. First and last lines are required; others may be optional. This method is only supported if **CapFixedOutput** is true. The Fiscal Printer state is set to FPTR_PS_FIXED_OUTPUT. 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 16.

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

Value	Meaning
E_BUSY	Cannot perform while output is in progress. (Only applies if AsyncMode is false.)
E_ILLEGAL	The Fiscal Printer does not support fixed output (see the CapFixedOutput property) or the lineNumber is invalid.
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not in the Fixed Output state. <i>ErrorCodeExtended</i> = EFPTR_COVER_OPEN: The Fiscal Printer cover is open. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_JRN_EMPTY: The journal station is out of paper. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_REC_EMPTY: The receipt station was specified but is out of paper. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted. (Only applies if AsyncMode is false.)

See Also **beginFixedOutput** Method, **endFixedOutput** Method

printNormal Method**Updated in Release 1.7**

Syntax **printNormal (station: *int32*, data: *string*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>station</i>	The Fiscal Printer station to be used. May be FPTR_S_RECEIPT, FPTR_S_JOURNAL, or FPTR_S_SLIP.
<i>data</i> ¹	The characters to be printed. May consist mostly of printable characters, escape sequences, carriage returns (13 decimal), and newline / line feeds (10 decimal) but in many cases these are not supported.

Remarks Performs non-fiscal printing. Prints *data* on the Fiscal Printer *station*.

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

Special character values within *data* are:

Value	Meaning
Newline / Line Feed (\n)	Print any data in the line buffer, and feed to the next print line. (A Carriage Return is not required in order to cause the line to be printed.)
Carriage Return (\r)	If a Carriage Return immediately precedes a Line Feed, or if the line buffer is empty, then it is ignored. Otherwise, the line buffer is printed and the Fiscal Printer does not feed to the next print line. On some Fiscal Printers, print without feed may be directly supported. On others, a print may always feed to the next line, in which case the Device will print the line buffer and perform a reverse line feed if supported. If the Fiscal Printer does not support either of these features, then Carriage Return acts like a Line Feed.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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. (Only applies if AsyncMode is false.)

¹. In the OPOS environment, the format of *data* depends upon the value of the **BinaryConversion** property. See **BinaryConversion** property on page A-28.

E_EXTENDED

ErrorCodeExtended = EFPTR_WRONG_STATE:
The Fiscal Printer is not currently in the Non-Fiscal state.

ErrorCodeExtended = EFPTR_COVER_OPEN:
The Fiscal Printer cover is open.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_JRN_EMPTY:
The journal station was specified but is out of paper.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_REC_EMPTY:
The receipt station was specified but is out of paper.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_SLP_EMPTY:
The slip station was specified, but a form is not inserted.
(Only applies if **AsyncMode** is false.)

See Also **beginNonFiscal** Method, **endNonFiscal** Method, **AsyncMode** Property.

printPeriodicTotalsReport Method

Syntax **printPeriodicTotalsReport (date1: *string*, date2: *string*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>date1</i>	Starting date of report to print.
<i>date2</i>	Ending date of report to print.

Remarks Prints a report of totals for a range of dates on the receipt.
 This method is always performed synchronously.

The dates are strings in the format “ddmmyyyhhmm”, where:

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

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer’s current state does not allow this state transition.
	<i>ErrorCodeExtended</i> = EFPTR_JRN_EMPTY: The journal station is out of paper.
	<i>ErrorCodeExtended</i> = EFPTR_REC_EMPTY: The receipt station is out of paper.
	<i>ErrorCodeExtended</i> = EFPTR_BAD_DATE: One of the date parameters is invalid.

printPowerLossReport Method

Syntax	printPowerLossReport (): void { raises-exception, use after open-claim-enable }
Remarks	Prints on the receipt a report of a power failure that resulted in a loss of data stored in the CMOS of the Fiscal Printer. This method is only supported if CapPowerLossReport is true. This method is always performed synchronously.
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	The Fiscal Printer does not support power loss reports (see the CapPowerLossReport property).
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer’s current state does not allow this state transition. <i>ErrorCodeExtended</i> = EFPTR_COVER_OPEN: The Fiscal Printer cover is open. <i>ErrorCodeExtended</i> = EFPTR_JRN_EMPTY: The journal station is out of paper. <i>ErrorCodeExtended</i> = EFPTR_REC_EMPTY: The receipt station is out of paper.

See Also **CapPowerLossReport** Property.

printRecCash Method**Added in Release 1.6**

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

Parameter	Description
-----------	-------------

<i>amount</i>	Amount to be incremented or decremented.
---------------	--

Remarks Prints a cash-in or cash-out receipt amount on the station defined by the **FiscalReceiptStation** property.

This method is only allowed if **CapFiscalReceiptType** is true and the **FiscalReceiptType** property is set to FPTR_RT_CASH_IN or FPTR_RT_CASH_OUT and the fiscal Fiscal Printer is in the Fiscal Receipt state.

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 16.

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

Value	Meaning
E_BUSY	Cannot perform while output is in progress. (Only applies if AsyncMode is false.)
E_ILLEGAL	The Fiscal Printer does not support this method.
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in the Fiscal Receipt state. <i>ErrorCodeExtended</i> = EFPTR_COVER_OPEN: The Fiscal Printer cover is open. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_JRN_EMPTY: The journal station is out of paper. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_REC_EMPTY: The receipt station is out of paper. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted. (Only applies if AsyncMode is false.)

See Also **beginFiscalReceipt** Method, **FiscalReceiptStation** Property,
FiscalReceiptType Property.

printRecItem Method**Updated in Release 1.6**

Syntax **printRecItem** (**description**: *string*, **price**: *currency*, **quantity**: *int32*, **vatInfo**: *int32*, **unitPrice**: *currency*, **unitName**: *string*):
 void { **raises-exception**, **use after open-claim-enable** }

Parameter	Description
<i>description</i>	Text describing the item sold.
<i>price</i>	Price of the line item.
<i>quantity</i>	Number of items. If zero, a single item is assumed.
<i>vatInfo</i>	VAT rate identifier or amount. If not used a zero is to be transferred.
<i>unitPrice</i>	Price of each item. If not used a zero is to be transferred.
<i>unitName</i>	Name of the unit i.e., “kg” or “ltr” or “pcs”. If not used an empty string (“”) is to be transferred

Remarks Prints a receipt item for a sold item on the station specified by the **FiscalReceiptStation** property. If the *quantity* parameter is zero, then a single item quantity will be assumed.

Minimum parameters are *description* and *price* or *description*, *price*, *quantity*, and *unitPrice*. Most countries require *quantity* and *vatInfo* and some countries also require *unitPrice* and *unitName*.

VatInfo parameter contains a VAT table identifier if **CapHasVatTable** is true. Otherwise, it contains a VAT amount.

If **CapPostPreLine** is true additional application specific lines defined by the **PostLine** and **PreLine** properties will be printed. After printing these lines **PostLine** and **PreLine** will be reset to an empty string.

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 16.

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

Value	Meaning
E_BUSY	Cannot perform while output is in progress. (Only applies if AsyncMode is false.)
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in the Fiscal Receipt state. <i>ErrorCodeExtended</i> = EFPTR_COVER_OPEN: The Fiscal Printer cover is open. (Only applies if AsyncMode is false.)

ErrorCodeExtended = EFPTR_JRN_EMPTY:

The journal station is out of paper.

(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_REC_EMPTY:

The receipt station is out of paper.

(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_SLP_EMPTY:

The slip station was specified, but a form is not inserted.

(Only applies if **AsyncMode** is false.)

ErrorCodeExtended =

EFPTR_BAD_ITEM_QUANTITY:

The quantity is invalid.

(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_BAD_PRICE:

The unit price is invalid.

(Only applies if **AsyncMode** is false.)

ErrorCodeExtended =

EFPTR_BAD_ITEM_DESCRIPTION:

The discount description is too long or contains a reserved word.

(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_BAD_VAT:

The VAT parameter is invalid.

(Only applies if **AsyncMode** is false.)

ErrorCodeExtended =

EFPTR_RECEIPT_TOTAL_OVERFLOW:

The receipt total has overflowed.

(Only applies if **AsyncMode** is false.)

See Also **beginFiscalReceipt** Method, **endFiscalReceipt** Method, **printRec...** Methods, **AmountDecimalPlaces** Property, **FiscalReceiptStation** Property, **PostLine** Property, **PreLine** Property.

printRecItemAdjustment Method**Updated in Release 1.6**

Syntax **printRecItemAdjustment (adjustmentType: *int32*, description: *string*, amount: *currency*, vatInfo: *int32*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>adjustmentType</i>	Type of adjustment. See below for values.
<i>description</i>	Text describing the adjustment.
<i>amount</i>	Amount of the adjustment.
<i>vatInfo</i>	VAT rate identifier or amount.

The *adjustmentType* parameter has one of the following values (*Note: If currency value, four decimal places are used*):

Value	Meaning
FPTR_AT_AMOUNT_DISCOUNT	Fixed amount discount. The <i>amount</i> parameter contains a currency value.
FPTR_AT_AMOUNT_SURCHARGE	Fixed amount surcharge. The <i>amount</i> parameter contains a currency value.
FPTR_AT_PERCENTAGE_DISCOUNT	Percentage discount. The <i>amount</i> parameter contains a percentage value.
FPTR_AT_PERCENTAGE_SURCHARGE	Percentage surcharge. The <i>amount</i> parameter contains a percentage value.

Remarks Applies and prints a discount or a surcharge to the last receipt item sold on the station specified by the **FiscalReceiptStation** property. This discount may be either a fixed currency amount or a percentage amount relating to the last item.

If **CapOrderAdjustmentFirst** is true, the method must be called before the corresponding **printRecItem** method. If **CapOrderAdjustmentFirst** is false, the method must be called after the **printRecItem**.

This discount/surcharge may be either a fixed currency amount or a percentage amount relating to the last item. If the discount amount is greater than the receipt subtotal, an error occurs since the subtotal can never be negative. In many countries discount operations cause the printing of a fixed line of text expressing the kind of operation that has been performed.

The *VatInfo* parameter contains a VAT table identifier if **CapHasVatTable** is true. Otherwise, it contains a VAT amount.

Fixed amount discounts/surcharges are only supported if the property **CapAmountAdjustment** is true. Percentage discounts are only supported if **CapPercentAdjustment** is true.

If **CapPostPreLine** is true an additional application specific line defined by the **PreLine** property will be printed. After printing this line **PreLine** will be reset to an empty string.

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 16.

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

Value	Meaning
E_BUSY	Cannot perform while output is in progress. (Only applies if AsyncMode is false.)
E_ILLEGAL	One of the following errors occurred: <ul style="list-style-type: none"> The Fiscal Printer does not support fixed amount adjustments (see the CapAmountAdjustment property). The Fiscal Printer does not support percentage discounts (see the CapPercentAdjustment property). The <i>adjustmentType</i> parameter is invalid.
E_EXTENDED	<p><i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in the Fiscal Receipt state.</p> <p><i>ErrorCodeExtended</i> = EFPTR_COVER_OPEN: The Fiscal Printer cover is open. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_JRN_EMPTY: The journal station is out of paper. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_REC_EMPTY: The receipt station is out of paper. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = FPTR_BAD_ITEM_AMOUNT: The discount amount is invalid. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_BAD_ITEM_DESCRIPTION: The discount description is too long or contains a reserved word. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_BAD_VAT: The VAT parameter is invalid. (Only applies if AsyncMode is false.)</p>

See Also **beginFiscalReceipt** Method, **endFiscalReceipt** Method, **printRec...** Methods, **AmountDecimalPlaces** Property, **FiscalReceiptStation** Property, **PreLine** Property.

printRecItemFuel Method**Added in Release 1.6**

Syntax **printRecItemFuel** (*description: string*, *price: currency*, *quantity: int32*, *vatInfo: int32*, *unitPrice: currency*, *unitName: string*, *specialTax: currency*, *specialTaxName: string*):
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>description</i>	Text describing the fuel product.
<i>price</i>	Price of the fuel item.
<i>quantity</i>	Number of items. If zero, a single item is assumed.
<i>vatInfo</i>	VAT rate identifier or amount. If not used a zero is to be transferred.
<i>unitPrice</i>	Price of the fuel item per volume.
<i>unitName</i>	Name of the volume unit, i.e., "ltr". If not used an empty string ("") is to be transferred
<i>specialTax</i>	Special tax amount, e.g., road tax. If not used a zero is to be transferred.
<i>specialTaxName</i>	Name of the special tax.

Remarks Prints a receipt fuel item on the station specified by the **FiscalReceiptStation** property. *vatInfo* parameter contains a VAT table identifier if **CapHasVatTable** is true. Otherwise, it contains a VAT amount.

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

Errors A `UposException` may be thrown when this method is invoked. For further information, see "Errors" on page 16.

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

Value	Meaning
E_BUSY	Cannot perform while output is in progress. (Only applies if AsyncMode is false.)
E_ILLEGAL	This method is not supported.

E_EXTENDED	<p><i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in the Fiscal Receipt state.</p> <p><i>ErrorCodeExtended</i> = EFPTR_COVER_OPEN: The Fiscal Printer cover is open. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_JRN_EMPTY: The journal station is out of paper. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_REC_EMPTY: The receipt station is out of paper. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_BAD_ITEM_QUANTITY: The quantity is invalid. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_BAD_PRICE: The unit price is invalid. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_BAD_ITEM_DESCRIPTION: The discount description is too long or contains a reserved word. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_BAD_VAT: The VAT parameter is invalid. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_RECEIPT_TOTAL_OVERFLOW: The receipt total has overflowed. (Only applies if AsyncMode is false.)</p>
------------	--

See Also **beginFiscalReceipt** Method, **FiscalReceiptStation** Property.

printRecItemFuelVoid Method**Added in Release 1.6**

Syntax **printRecItemFuelVoid** (**description**: *string*, **price**: *currency*, **vatInfo**: *int32*, **specialTax**: *currency*):
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>description</i>	Text describing the fuel product.
<i>price</i>	Price of the fuel item. If not used a zero is to be transferred.
<i>vatInfo</i>	VAT rate identifier or amount. If not used a zero is to be transferred.
<i>specialTax</i>	Special tax amount, e.g., road tax. If not used a zero is to be transferred.

Remarks Called to void a fuel item on the station specified by the **FiscalReceiptStation** property.

If **CapOnlyVoidLastItem** is true, only the last fuel item transferred to the Fiscal Printer can be voided.

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 16.

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

Value	Meaning
E_BUSY	Cannot perform while output is in progress. (Only applies if AsyncMode is false.)
E_ILLEGAL	This method is not supported.

E_EXTENDED	<p><i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in the Fiscal Receipt state.</p> <p><i>ErrorCodeExtended</i> = EFPTR_COVER_OPEN: The Fiscal Printer cover is open. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_JRN_EMPTY: The journal station is out of paper. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_REC_EMPTY: The receipt station is out of paper. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_BAD_PRICE: The price is invalid. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_BAD_ITEM_DESCRIPTION: The discount description is too long or contains a reserved word. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_BAD_VAT: The VAT parameter is invalid. (Only applies if AsyncMode is false.)</p>
------------	---

See Also **beginFiscalReceipt** Method, **endFiscalReceipt** Method,
printRecItemFuel Method, **CapOnlyVoidLastItem** Property,
FiscalReceiptStation Property.

printRecMessage Method**Updated in Release 1.6**

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

Parameter	Description
-----------	-------------

<i>message</i>	Text message to print.
----------------	------------------------

Remarks Prints a message on the fiscal receipt on the station specified by the **FiscalReceiptStation** property. The length of an individual message is limited to the number of characters given in the **MessageLength** property. The kind of message to be printed is defined by the **MessageType** property.

This method is only supported if **CapAdditionalLines** is true. This method is only supported when the Fiscal Printer is in the Fiscal Receipt Ending state.

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

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
-------	---------

E_BUSY	Cannot perform while output is in progress. (Only applies if AsyncMode is false.)
--------	---

E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not in the Fiscal Receipt Ending state. <i>ErrorCodeExtended</i> = EFPTR_COVER_OPEN: The Fiscal Printer cover is open. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_JRN_EMPTY: The journal station is out of paper. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_REC_EMPTY: The receipt station is out of paper. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_BAD_ITEM_DESCRIPTION: The message is too long or contains a reserved word. (Only applies if AsyncMode is false.)
------------	--

See Also **beginFiscalReceipt** Method, **endFiscalReceipt** Method, **printRec...** Methods, **CapAdditionalLines** Property, **FiscalReceiptStation** Property, **MessageLength** Property, **MessageType** Property.

printRecNotPaid Method**Updated in Release 1.6**

Syntax	printRecNotPaid (description: <i>string</i>, amount: <i>currency</i>): void { raises-exception, use after open-claim-enable }						
	<table> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><i>description</i></td> <td>Text describing the not paid amount.</td> </tr> <tr> <td><i>amount</i></td> <td>Amount not paid.</td> </tr> </tbody> </table>	Parameter	Description	<i>description</i>	Text describing the not paid amount.	<i>amount</i>	Amount not paid.
Parameter	Description						
<i>description</i>	Text describing the not paid amount.						
<i>amount</i>	Amount not paid.						
Remarks	<p>Indicates a part of the receipt's total to not be paid.</p> <p>Some fixed text, along with the <i>description</i>, will be printed on the station defined by the FiscalReceiptStation property to indicate that part of the receipt total has not been paid. This method is only supported if CapAmountNotPaid is true. If this method is successful, the PrinterState property will remain in FPTR_PS_FISCAL_RECEIPT_TOTAL state or change to the value FPTR_PS_FISCAL_RECEIPT_ENDING depending upon whether the entire receipt total is now accounted for or not. This method is performed synchronously if AsyncMode is false, and asynchronously if AsyncMode is true.</p>						
Errors	<p>A UposException may be thrown when this method is invoked. For further information, see "Errors" on page 16.</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_BUSY</td> <td>Cannot perform while output is in progress. (Only applies if AsyncMode is false.)</td> </tr> <tr> <td>E_EXTENDED</td> <td> <i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in either the Fiscal Receipt or Fiscal Receipt Total state. <i>ErrorCodeExtended</i> = EFPTR_COVER_OPEN: The Fiscal Printer cover is open. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_JRN_EMPTY: The journal station is out of paper. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_REC_EMPTY: The receipt station is out of paper. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_BAD_ITEM_DESCRIPTION: The <i>description</i> is too long or contains a reserved word. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_BAD_ITEM_AMOUNT: The <i>amount</i> is invalid. (Only applies if AsyncMode is false.) </td> </tr> </tbody> </table>	Value	Meaning	E_BUSY	Cannot perform while output is in progress. (Only applies if AsyncMode is false.)	E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE : The Fiscal Printer is not currently in either the Fiscal Receipt or Fiscal Receipt Total state. <i>ErrorCodeExtended</i> = EFPTR_COVER_OPEN : The Fiscal Printer cover is open. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_JRN_EMPTY : The journal station is out of paper. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_REC_EMPTY : The receipt station is out of paper. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_SLP_EMPTY : The slip station was specified, but a form is not inserted. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_BAD_ITEM_DESCRIPTION : The <i>description</i> is too long or contains a reserved word. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_BAD_ITEM_AMOUNT : The <i>amount</i> is invalid. (Only applies if AsyncMode is false.)
Value	Meaning						
E_BUSY	Cannot perform while output is in progress. (Only applies if AsyncMode is false.)						
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE : The Fiscal Printer is not currently in either the Fiscal Receipt or Fiscal Receipt Total state. <i>ErrorCodeExtended</i> = EFPTR_COVER_OPEN : The Fiscal Printer cover is open. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_JRN_EMPTY : The journal station is out of paper. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_REC_EMPTY : The receipt station is out of paper. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_SLP_EMPTY : The slip station was specified, but a form is not inserted. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_BAD_ITEM_DESCRIPTION : The <i>description</i> is too long or contains a reserved word. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_BAD_ITEM_AMOUNT : The <i>amount</i> is invalid. (Only applies if AsyncMode is false.)						
See Also	beginFiscalReceipt Method, endFiscalReceipt Method, printRec... Methods, AmountDecimalPlaces Property, FiscalReceiptStation Property.						

printRecPackageAdjustment Method**Added in Release 1.6**

Syntax **printRecPackageAdjustment (adjustmentType: *int32*,
description: *string*, vatAdjustment: *string*):
void { raises-exception, use after open-claim-enable }**

Parameter	Description
<i>adjustmentType</i>	Type of adjustment. See below for values.
<i>description</i>	Text describing the adjustment.
<i>vatAdjustment</i>	String containing a list of adjustment(s) for different Vat(s).

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

Value	Meaning
FPTR_AT_DISCOUNT	Discount.
FPTR_AT_SURCHARGE	Surcharge.

The *vatAdjustment* parameter consists of ASCII numeric semicolon delimited pairs of values which denote each the VAT identifier of the package item to be adjusted and adjustment amount, separated by a comma.

The number of pairs is delimited by the **NumVatRates** property.

Remarks Called to give an adjustment for a package of some items booked before. This adjustment (discount/surcharge) may be either a fixed currency amount or a percentage amount relating to items combined to an adjustment package.

Each item of the package must be transferred before.

Fixed amount adjustments are only supported if **CapPackageAdjustment** is true.

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 16.

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

Value	Meaning
E_BUSY	Cannot perform while output is in progress. (Only applies if AsyncMode is false.)
E_ILLEGAL	The Fiscal Printer does not support package adjustments (see the CapPackageAdjustment property), or the <i>adjustmentType</i> parameter is invalid.

E_EXTENDED

ErrorCodeExtended = EFPTR_WRONG_STATE:
The Fiscal Printer is not currently in the Fiscal Receipt state.

ErrorCodeExtended = EFPTR_COVER_OPEN:
The Fiscal Printer cover is open.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_JRN_EMPTY:
The journal station is out of paper.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_REC_EMPTY:
The receipt station is out of paper.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_SLP_EMPTY:
The slip station was specified, but a form is not inserted.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended =
EFPTR_BAD_ITEM_DESCRIPTION:
The *description* is too long or contains a reserved word.
(Only applies if **AsyncMode** is false.)

See Also **printRecPackageAdjustVoid** Method, **CapPackageAdjustment** Property.

E_EXTENDED

ErrorCodeExtended = EFPTR_WRONG_STATE:
The Fiscal Printer is not currently in the Fiscal Receipt state.

ErrorCodeExtended = EFPTR_COVER_OPEN:
The Fiscal Printer cover is open.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_JRN_EMPTY:
The journal station is out of paper.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_REC_EMPTY:
The receipt station is out of paper.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_SLP_EMPTY:
The slip station was specified, but a form is not inserted.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended =
EFPTR_BAD_ITEM_DESCRIPTION:
The *description* is too long or contains a reserved word.
(Only applies if **AsyncMode** is false.)

See Also **printRecPackageAdjustment** Method, **CapPackageAdjustment** Property, **PreLine** Property.

printRecRefund Method**Updated in Release 1.6**

Syntax **printRecRefund (description: *string*, amount: *currency*, vatInfo: *int32*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>description</i>	Text describing the refund.
<i>amount</i>	Amount of the refund.
<i>vatInfo</i>	VAT rate identifier or amount.

Remarks Processes a refund. The *amount* is positive, but it is printed as a negative number and the totals registers are decremented.

Some fixed text, along with the *description*, will be printed on the station defined by the **FiscalReceiptStation** property to indicate that a refund has occurred.

The *vatInfo* parameter contains a VAT table identifier if **CapHasVatTable** is true. Otherwise it, contains a VAT amount.

If **CapPostPreLine** is true an additional application specific line defined by the **PreLine** property will be printed. After printing this line **PreLine** will be reset to an empty string.

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 16.

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

Value	Meaning
E_BUSY	Cannot perform while output is in progress. (Only applies if AsyncMode is false.)

E_EXTENDED *ErrorCodeExtended* = EFPTR_WRONG_STATE:
The Fiscal Printer is not currently in the Fiscal Receipt state.

ErrorCodeExtended = EFPTR_COVER_OPEN:
The Fiscal Printer cover is open.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_JRN_EMPTY:
The journal station is out of paper.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_REC_EMPTY:
The receipt station is out of paper.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_SLP_EMPTY:
The slip station was specified, but a form is not inserted.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended =
EFPTR_BAD_ITEM_DESCRIPTION:
The *description* is too long or contains a reserved word.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended =
EFPTR_BAD_ITEM_AMOUNT:
The *amount* is invalid.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_BAD_VAT:
The VAT information is invalid.
(Only applies if **AsyncMode** is false.)

See Also **beginFiscalReceipt** Method, **endFiscalReceipt** Method, **printRec...** Methods, **AmountDecimalPlaces** Property, **FiscalReceiptStation** Property, **PreLine** Property.

E_EXTENDED

ErrorCodeExtended = EFPTR_WRONG_STATE:
The Fiscal Printer is not currently in the Fiscal Receipt state.

ErrorCodeExtended = EFPTR_COVER_OPEN:
The Fiscal Printer cover is open.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_JRN_EMPTY:
The journal station is out of paper.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_REC_EMPTY:
The receipt station is out of paper.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_SLP_EMPTY:
The slip station was specified, but a form is not inserted.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended =
EFPTR_BAD_ITEM_DESCRIPTION:
The *description* is too long or contains a reserved word.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended =
EFPTR_BAD_ITEM_AMOUNT:
The *amount* is invalid.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_BAD_VAT:
The VAT information is invalid.
(Only applies if **AsyncMode** is false.)

See Also **printRecRefund** Method, **FiscalReceiptStation** Property.

printRecSubtotal Method**Updated in Release 1.6**

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

Parameter	Description
------------------	--------------------

<i>amount</i>	Amount of the subtotal.
---------------	-------------------------

Remarks Checks and prints the current receipt subtotal on the station defined by the **FiscalReceiptStation** property.

If **CapCheckTotal** is true, the *amount* is compared to the subtotal calculated by the Fiscal Printer. If the subtotals match, the subtotal is printed on the station defined by the **FiscalReceiptStation** property. If the results do not match, the receipt is automatically canceled. If **CapCheckTotal** is false, then the subtotal is printed on the station defined by the **FiscalReceiptStation** property and the parameter is never compared to the subtotal computed by the Fiscal Printer.

If **CapPostPreLine** is true an additional application specific line defined by the **PostLine** property will be printed. After printing this line **PostLine** will be reset to an empty string.

If this method compares the application's subtotal with the Fiscal Printer's subtotal and they do not match, the **PrinterState** property will be changed to FPTR_PS_FISCAL_RECEIPT_ENDING.

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 16.

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

Value	Meaning
--------------	----------------

E_BUSY	Cannot perform while output is in progress. (Only applies if AsyncMode is false.)
--------	--

E_EXTENDED *ErrorCodeExtended* = EFPTR_WRONG_STATE:
The Fiscal Printer is not currently in the Fiscal Receipt state.

ErrorCodeExtended = EFPTR_COVER_OPEN:
The Fiscal Printer cover is open.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_JRN_EMPTY:
The journal station is out of paper.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_REC_EMPTY:
The receipt station is out of paper.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_SLP_EMPTY:
The slip station was specified, but a form is not inserted.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended =
EFPTR_BAD_ITEM_AMOUNT:
The subtotal from the application does not match the
subtotal computed by the Fiscal Printer.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_NEGATIVE_TOTAL:
The total computed by the Fiscal Printer is less than
zero.
(Only applies if **AsyncMode** is false.)

See Also **beginFiscalReceipt** Method, **endFiscalReceipt** Method, **printRec...** Methods,
AmountDecimalPlaces Property, **FiscalReceiptStation** Property,
PostLine Property.

printRecSubtotalAdjustment Method**Updated in Release 1.6**

Syntax **printRecSubtotalAdjustment (adjustmentType: *int32*,
description: *string*, amount: *currency*):
void { raises-exception, use after open-claim-enable }**

Parameter	Description
<i>adjustmentType</i>	Type of adjustment. See below for values.
<i>description</i>	Text describing the discount or surcharge.
<i>amount</i>	Amount of the adjustment (discount or surcharge).

The *adjustmentType* parameter has one of the following values (*Note: If currency value, four decimal places are used*):

Value	Meaning
FPTR_AT_AMOUNT_DISCOUNT	Fixed amount discount. The <i>amount</i> parameter contains a currency value.
FPTR_AT_AMOUNT_SURCHARGE	Fixed amount surcharge. The <i>amount</i> parameter contains a currency value.
FPTR_AT_PERCENTAGE_DISCOUNT	Percentage discount. The <i>amount</i> parameter contains a percentage value.
FPTR_AT_PERCENTAGE_SURCHARGE	Percentage surcharge. The <i>amount</i> parameter contains a percentage value.

Remarks Applies and prints a discount/surcharge to the current receipt subtotal on the station defined by the **FiscalReceiptStation** property. This discount/surcharge may be either a fixed currency amount or a percentage amount relating to the current receipt subtotal.

If the discount/surcharge amount is greater than the receipt subtotal, an error occurs since the subtotal can never be negative.

In many countries discount/surcharge operations cause the printing of a fixed line of text expressing the kind of operation that has been performed.

Fixed amount discounts are only supported if **CapSubAmountAdjustment** is true. Percentage discounts are only supported if **CapSubPercentAdjustment** is true.

If **CapPostPreLine** is true an additional application specific line defined by the **PreLine** property will be printed. After printing this line **PreLine** will be reset to an empty string.

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 16.

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

Value	Meaning
E_BUSY	Cannot perform while output is in progress. (Only applies if AsyncMode is false.)
E_ILLEGAL	One of the following errors occurred: <ul style="list-style-type: none"> • Fixed amount discounts are not supported (see the CapSubAmountAdjustment property). • Percentage discounts are not supported (see the CapSubPercentAdjustment property). • The <i>adjustmentType</i> parameter is invalid.
E_EXTENDED	<p><i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in the Fiscal Receipt state.</p> <p><i>ErrorCodeExtended</i> = EFPTR_COVER_OPEN: The Fiscal Printer cover is open. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_JRN_EMPTY: The journal station is out of paper. (Only applies AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_REC_EMPTY: The receipt station is out of paper. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_BAD_ITEM_AMOUNT: The discount <i>amount</i> is invalid. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_BAD_ITEM_DESCRIPTION: The discount <i>description</i> is too long or contains a reserved word. (Only applies if AsyncMode is false.)</p>

See Also **beginFiscalReceipt** Method, **endFiscalReceipt** Method, **printRec...** Methods, **AmountDecimalPlaces** Property, **FiscalReceiptStation** Property, **PreLine** Property.

Errors A `UposException` may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
<code>E_BUSY</code>	Cannot perform while output is in progress. (Only applies if <code>AsyncMode</code> is false.)
<code>E_ILLEGAL</code>	One of the following errors occurred: <ul style="list-style-type: none"> Fixed amount discounts are not supported (see the <code>CapSubAmountAdjustment</code> property). Percentage discounts are not supported (see the <code>CapSubPercentAdjustment</code> property). The <code>adjustmentType</code> parameter is invalid.
<code>E_EXTENDED</code>	<p><code>ErrorCodeExtended = EFPTR_WRONG_STATE:</code> The Fiscal Printer is not currently in the Fiscal Receipt state.</p> <p><code>ErrorCodeExtended = EFPTR_COVER_OPEN:</code> The Fiscal Printer cover is open. (Only applies if <code>AsyncMode</code> is false.)</p> <p><code>ErrorCodeExtended = EFPTR_JRN_EMPTY:</code> The journal station is out of paper. (Only applies if <code>AsyncMode</code> is false.)</p> <p><code>ErrorCodeExtended = EFPTR_REC_EMPTY:</code> The receipt station is out of paper. (Only applies if <code>AsyncMode</code> is false.)</p> <p><code>ErrorCodeExtended = EFPTR_SLP_EMPTY:</code> The slip station was specified, but a form is not inserted. (Only applies if <code>AsyncMode</code> is false.)</p> <p><code>ErrorCodeExtended = EFPTR_BAD_ITEM_AMOUNT:</code> The discount <code>amount</code> is invalid. (Only applies if <code>AsyncMode</code> is false.)</p>

See Also `beginFiscalReceipt` Method, `endFiscalReceipt` Method, `printRec...` Methods, `AmountDecimalPlaces` Property, `FiscalReceiptStation` Property, `PreLine` Property.

printRecTaxID Method**Added in Release 1.6**

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

Parameter	Description
<i>taxId</i>	Customer identification with identification characters and tax number.

Remarks Called to print the customers tax identification on the station defined by the **FiscalReceiptStation** property.

This method is only supported when the Fiscal Printer is in the Fiscal Receipt Ending state.

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

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_BUSY	Cannot perform while output is in progress. (Only applies if AsyncMode is false.)
E_ILLEGAL	The Fiscal Printer does not support printing tax identifications.
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in the Fiscal Receipt Ending state. <i>ErrorCodeExtended</i> = EFPTR_COVER_OPEN: The Fiscal Printer cover is open. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_JRN_EMPTY: The journal station is out of paper. (Only applies AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_REC_EMPTY: The receipt station is out of paper. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted. (Only applies if AsyncMode is false.)

See Also **FiscalReceiptStation** Property.

printRecTotal Method**Updated in Release 1.6**

Syntax **printRecTotal (total: *int32*, payment: *int32*, description: *string*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>total</i>	Application computed receipt total.
<i>payment</i>	Amount of payment tendered.
<i>description</i>	Text description of the payment or the index of a predefined payment description.

Remarks Checks and prints the current receipt total on the station defined by the **FiscalReceiptStation** property and to tender a payment.

If **CapCheckTotal** is true, the *total* is compared to the total calculated by the Fiscal Printer. If the totals match, the total is printed on both the receipt and journal along with some fixed text. If the results do not match, the receipt is automatically canceled. If **CapCheckTotal** is false, then the total is printed on the receipt and journal and the parameter is never compared to the total computed by the Fiscal Printer.

If **CapPredefinedPaymentLines** is true, then the *description* parameter contains the index of one of the Fiscal Printer's predefined payment descriptions. The index is typically a single character of the alphabet. The set of allowed values for this index is to be described in the description of the service and stored in the **PredefinedPaymentLines** property.

If *payment* = *total*, a line containing the *description* and *payment* is printed. The **PrinterState** property will be set to FPTR_PS_FISCAL_RECEIPT_ENDING.

If *payment* > *total*, a line containing the *description* and *payment* is printed followed by a second line containing the change due. If **CapChangeDue** property is true, a description for the change due defined by the **ChangeDue** property is printed as the second line. The **PrinterState** property will be set to FPTR_PS_FISCAL_RECEIPT_ENDING.

If *payment* < *total*, a line containing the *description* and *payment* is printed. Since the entire receipt total has not yet been tendered, the **PrinterState** property will be set to FPTR_PS_FISCAL_RECEIPT_TOTAL.

If *payment* = 0, no line containing the *description* and *payment* is printed. The **PrinterState** property will be set to FPTR_PS_FISCAL_RECEIPT_TOTAL.

If **CapAdditionalLines** is false, then receipt trailer lines, fiscal logotype and receipt cut are executed after the last total line, whenever receipt's total became equal to the payment from the application. Otherwise these lines are printed calling the **endFiscalReceipt** method.

If **CapPostPreLine** is true an additional application specific line defined by the **PostLine** property will be printed. After printing this line **PostLine** will be reset to an empty string.

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 16.

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

Value	Meaning
E_BUSY	Cannot perform while output is in progress. (Only applies if AsyncMode is false.)
E_EXTENDED	<p><i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in the Fiscal Receipt state.</p> <p><i>ErrorCodeExtended</i> = EFPTR_COVER_OPEN: The Fiscal Printer cover is open. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_JRN_EMPTY: The journal station is out of paper. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_REC_EMPTY: The receipt station is out of paper. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_BAD_ITEM_AMOUNT: <ul style="list-style-type: none"> • The application computed total does not match the Fiscal Printer computed total, or • the <i>total</i> parameter is invalid, or • the <i>payment</i> parameter is invalid (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_BAD_ITEM_DESCRIPTION: The <i>description</i> is too long or contains a reserved word. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_NEGATIVE_TOTAL: The computed total is less than zero. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_WORD_NOT_ALLOWED: The description contains the reserved word.</p>

See Also **beginFiscalReceipt** Method, **endFiscalReceipt** Method, **printRec...** Methods, **PredefinedPaymentLines** Property, **AmountDecimalPlaces** Property, **ChangeDue** Property, **FiscalReceiptStation** Property, **PostLine** Property.

printRecVoid Method**Updated in Release 1.6**

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

Parameter	Description
<i>description</i>	Text describing the void.

Remarks Cancels the current receipt.

The receipt is annulled but it is not physically canceled from the Fiscal Printer's fiscal memory since fiscal receipts are printed with an increasing serial number and totals are accumulated in registers. When a receipt is canceled, its subtotal is subtracted from the totals registers, but it is added to the canceled receipt register.

Some fixed text, along with the *description*, will be printed on the station defined by the **FiscalReceiptStation** property to indicate that the receipt has been canceled.

Normally only a receipt with at least one transaction can be voided. If **CapEmptyReceiptsVoidable** is true also an empty receipt (only the **beginFiscalReceipt** method was called) can be voided.

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

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 16.

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

Value	Meaning
E_BUSY	Cannot perform while output is in progress. (Only applies if AsyncMode is false.)

E_EXTENDED

ErrorCodeExtended = EFPTR_WRONG_STATE:
The Fiscal Printer is not currently in the Fiscal Receipt state.

ErrorCodeExtended = EFPTR_COVER_OPEN:
The Fiscal Printer cover is open.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_JRN_EMPTY:
The journal station is out of paper.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_REC_EMPTY:
The receipt station is out of paper.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended = EFPTR_SLP_EMPTY:
The slip station was specified, but a form is not inserted.
(Only applies if **AsyncMode** is false.)

ErrorCodeExtended =
EFPTR_BAD_ITEM_DESCRIPTION:
The description is too long or contains a reserved word.
(Only applies if **AsyncMode** is false.)

See Also **beginFiscalReceipt** Method, **endFiscalReceipt** Method, **printRec...** Methods
CapEmptyReceiptIsVoidable Property, **FiscalReceiptStation** Property.

printRecVoidItem Method**Updated in Release 1.6**

Syntax **printRecVoidItem** (**description**: *string*, **amount**: *currency*,
quantity: *int32*, **adjustmentType**: *int32*,
adjustment: *currency*, **vatInfo**: *int32*):
void { **raises-exception**, **use after open-claim-enable** }

Parameter	Description
<i>description</i>	Text description of the item void.
<i>amount</i>	Amount of item to be voided.
<i>quantity</i>	Quantity of item to be voided.
<i>adjustmentType</i>	Type of adjustment. See below for values.
<i>adjustment</i>	Amount of the adjustment (discount or surcharge).
<i>vatInfo</i>	VAT rate identifier or amount.

The *adjustmentType* parameter has one of the following values (*Note: If currency value, four decimal places are used*):

Value	Meaning
FPTR_AT_AMOUNT_DISCOUNT	Fixed amount discount. The <i>adjustment</i> parameter contains a currency value.
FPTR_AT_AMOUNT_SURCHARGE	Fixed amount surcharge. The <i>adjustment</i> parameter contains a currency value.
FPTR_AT_PERCENTAGE_DISCOUNT	Percentage discount. The <i>adjustment</i> parameter contains a percentage value.
FPTR_AT_PERCENTAGE_SURCHARGE	Percentage surcharge. The <i>adjustment</i> parameter contains a percentage value.

Remarks Cancels an item that has been added to the receipt and prints a void description on the station defined by the **FiscalReceiptStation** property.

amount is a positive number, it will be printed as a negative and will be decremented from the totals registers.

The *vatInfo* parameter contains a VAT table identifier if **CapHasVatTable** is true. Otherwise, it contains a VAT amount. Fixed amount discounts/surcharges are only supported if **CapAmountAdjustment** is true. Percentage discounts are only supported if **CapPercentAdjustment** is true.

If **CapOnlyVoidLastItem** is true, only the last item transferred to the Fiscal Printer can be voided.

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 16.

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

Value	Meaning
E_BUSY	Cannot perform while output is in progress. (Only applies if AsyncMode is false.)
E_ILLEGAL	One of the following errors occurred: <ul style="list-style-type: none"> Fixed amount adjustments are not supported (see the CapAmountAdjustment property), or Percentage discounts are not supported (see the CapPercentAdjustment property), or The <i>adjustmentType</i> parameter is invalid.
E_EXTENDED	<p><i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in the Fiscal Receipt state.</p> <p><i>ErrorCodeExtended</i> = EFPTR_COVER_OPEN: The Fiscal Printer cover is open. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_JRN_EMPTY: The journal station is out of paper. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_REC_EMPTY: The receipt station is out of paper. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_BAD_ITEM_AMOUNT: The <i>amount</i> is invalid. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_BAD_ITEM_QUANTITY: The <i>quantity</i> is invalid. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_BAD_VAT: The VAT information is invalid. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_BAD_ITEM_DESCRIPTION: The <i>description</i> is too long or contains a reserved word. (Only applies if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EFPTR_NEGATIVE_TOTAL: The computed total is less than zero. (Only applies if AsyncMode is false.)</p>

See Also **beginFiscalReceipt** Method, **endFiscalReceipt** Method, **printRec...** Methods, **CapOnlyVoidLastItem** Property, **AmountDecimalPlaces** Property, **FiscalReceiptStation** Property.

printReport Method

Syntax **printReport (reportType: *int32*, startNum: *string*, endNum: *string*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>reportType</i>	The kind of report to print.
<i>startNum</i>	ASCII string identifying the starting record in Fiscal Printer memory from which to begin printing
<i>endNum</i>	ASCII string identifying the final record in Fiscal Printer memory at which printing is to end. See <i>reportType</i> table below to find out the exact meaning of this parameter.

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

Value	Meaning
FPTR_RT_ORDINAL	Prints a report between two Z reports. If both <i>startNum</i> and <i>endNum</i> are valid and <i>endNum</i> > <i>startNum</i> , then a report of the period between <i>startNum</i> and <i>endNum</i> will be printed. If <i>startNum</i> is valid and <i>endNum</i> is zero, then a report of relating only to <i>startNum</i> will be printed.
FPTR_RT_DATE	Prints a report between two dates. The dates are strings in the format “ddmmyyyhhmm”, where:
dd	day of the month (01 - 31)
mm	month (01 - 12)
yyyy	year (1997- ...)
hh	hour (00-23)
mm	minutes (00-59)

Remarks Prints a report of the fiscal EPROM contents on the receipt that occurred between two end points.

This method is always performed synchronously.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_BUSY	Cannot perform while output is in progress.
E_ILLEGAL	One of the following errors occurred: <ul style="list-style-type: none"> • The <i>reportType</i> parameter is invalid, or • One or both of <i>startNum</i> and <i>endNum</i> are invalid, or • <i>startNum</i> > <i>endNum</i>.

E_EXTENDED	<p><i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer's current state does not allow this state transition.</p> <p><i>ErrorCodeExtended</i> = EFPTR_COVER_OPEN: The Fiscal Printer cover is open.</p> <p><i>ErrorCodeExtended</i> = EFPTR_JRN_EMPTY: The journal station is out of paper.</p> <p><i>ErrorCodeExtended</i> = EFPTR_REC_EMPTY: The receipt station is out of paper.</p>
------------	---

printXReport Method

Syntax	printXReport (): void { raises-exception, use after open-claim-enable }						
Remarks	<p>Prints a report of all the daily fiscal activities on the receipt. No data will be written to the fiscal EPROM as a result of this method invocation.</p> <p>This method is only supported if CapXReport is true. This method is always performed synchronously.</p>						
Errors	<p>A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table border="1"> <thead> <tr> <th style="text-align: left;">Value</th> <th style="text-align: left;">Meaning</th> </tr> </thead> <tbody> <tr> <td>E_ILLEGAL</td> <td>The Fiscal Printer does not support X reports (see the CapXReport property).</td> </tr> <tr> <td>E_EXTENDED</td> <td> <p><i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer’s current state does not allow this state transition.</p> <p><i>ErrorCodeExtended</i> = EFPTR_COVER_OPEN: The Fiscal Printer cover is open.</p> <p><i>ErrorCodeExtended</i> = EFPTR_JRN_EMPTY: The journal station is out of paper.</p> <p><i>ErrorCodeExtended</i> = EFPTR_REC_EMPTY: The receipt station is out of paper.</p> </td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	The Fiscal Printer does not support X reports (see the CapXReport property).	E_EXTENDED	<p><i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer’s current state does not allow this state transition.</p> <p><i>ErrorCodeExtended</i> = EFPTR_COVER_OPEN: The Fiscal Printer cover is open.</p> <p><i>ErrorCodeExtended</i> = EFPTR_JRN_EMPTY: The journal station is out of paper.</p> <p><i>ErrorCodeExtended</i> = EFPTR_REC_EMPTY: The receipt station is out of paper.</p>
Value	Meaning						
E_ILLEGAL	The Fiscal Printer does not support X reports (see the CapXReport property).						
E_EXTENDED	<p><i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer’s current state does not allow this state transition.</p> <p><i>ErrorCodeExtended</i> = EFPTR_COVER_OPEN: The Fiscal Printer cover is open.</p> <p><i>ErrorCodeExtended</i> = EFPTR_JRN_EMPTY: The journal station is out of paper.</p> <p><i>ErrorCodeExtended</i> = EFPTR_REC_EMPTY: The receipt station is out of paper.</p>						
See Also	CapXReport Property.						

printZReport Method**Updated in Release 1.6**

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

Remarks Prints a report of all the daily fiscal activities on the receipt. Data will be written to the fiscal EPROM as a result of this method invocation.

Since running **printZReport** is implicitly a fiscal end of day function, the **DayOpened** property will be set to false.

This method is always performed synchronously.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer’s current state does not allow this state transition.
	<i>ErrorCodeExtended</i> = EFPTR_COVER_OPEN: The Fiscal Printer cover is open.
	<i>ErrorCodeExtended</i> = EFPTR_JRN_EMPTY: The journal station is out of paper.
	<i>ErrorCodeExtended</i> = EFPTR_REC_EMPTY: The receipt station is out of paper.

See Also **beginFiscalDocument** Method, **beginFiscalReceipt** Method, **DayOpened** Property.

resetPrinter Method

Syntax	resetPrinter (): void { raises-exception, use after open-claim-enable }
Remarks	<p>Forces the Fiscal Printer to return to Monitor state. This forces any interrupted operations to be canceled and closed. This method must be invoked when the Fiscal Printer is not in a Monitor state after a successful call to the claim method and successful setting of the DeviceEnabled property to true. This typically happens if a power failures occurs during a fiscal operation.</p> <p>Calling this method does not close the Fiscal Printer, i.e., does not force a Z report to be printed.</p> <p>The Device will handle this command as follows:</p> <ul style="list-style-type: none">• If the Fiscal Printer was in either Fiscal Receipt, Fiscal Receipt Total or Fiscal Receipt Ending state, the receipt will be ended without updating any registers.• If the Fiscal Printer was in a non-fiscal state, the Fiscal Printer will exit that state.• If the Fiscal Printer was in the training state, the Fiscal Printer will exit the training state. <p>This method is always performed synchronously.</p>
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

setCurrency Method**Added in Release 1.6**

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

Parameter	Description
-----------	-------------

<i>newCurrency</i>	The new currency.
--------------------	-------------------

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

Value	Meaning
-------	---------

FPTR_SC_EURO	Change to the EURO currency.
--------------	------------------------------

Remarks Called to change to a new currency, e.g., EURO.

This method is only supported if **CapSetCurrency** is true and can only be called while **DayOpened** is false.

The actual currency is kept in the **ActualCurrency** property.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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"> • The Fiscal Printer does not support this method (see the CapSetCurrency property), or • The Fiscal Printer has already begun the fiscal day (see the DayOpened property), or • the specified <i>newCurrency</i> value is not valid.
-----------	---

See Also **ActualCurrency** Property, **CapSetCurrency** Property, **DayOpened** Property.

setDate Method

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

Parameter	Description
<i>date</i>	Date and time as a string.

Remarks Sets the Fiscal Printer's date and time.
 The date and time is passed as a string in the format "ddmmyyyhhmm", where:

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

This method can only be called while **DayOpened** is false.

Errors A UposException may be thrown when this method is invoked. For further information, see "Errors" on page 16.

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

Value	Meaning
E_ILLEGAL	The Fiscal Printer has already begun the fiscal day (see the DayOpened property).
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_BAD_DATE: One of the entries of the <i>date</i> parameters is invalid.

See Also **DayOpened** Property.

setPOSID Method

Syntax **setPOSID (POSID: *string*, cashierID: *string*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>POSID</i>	Identifier for the POS system.
<i>cashierID</i>	Identifier of the current cashier.

Remarks Sets the POS and cashier identifiers. These values will be printed when each fiscal receipt is closed.

This method is only supported if **CapSetPOSID** is true. This method can only be called while **DayOpened** is false.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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"> • The Fiscal Printer does not support setting the POS identifier (see the CapSetPOSID property), or • The printer has already begun the fiscal day (see the DayOpened property), or • Either the <i>POSID</i> or <i>cashierID</i> parameter is invalid.

See Also **CapSetPOSID** Property, **DayOpened** Property.

setStoreFiscalID Method

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

Parameter	Description
------------------	--------------------

<i>ID</i>	Fiscal identifier.
-----------	--------------------

Remarks Sets the store fiscal ID. This value is retained by the Fiscal Printer even after power failures. This *ID* is automatically printed by the Fiscal Printer after the fiscal receipt header lines.

This method is only supported if **CapSetStoreFiscalID** is true. This method can only be called while **DayOpened** is false.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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"> • The Fiscal Printer does not support setting the store fiscal identifier (see the CapSetStoreFiscalID property), or • The Fiscal Printer has already begun the fiscal day (see the DayOpened property), or • The <i>ID</i> parameter was invalid.
-----------	--

See Also **CapSetStoreFiscalID** Property, **DayOpened** Property.

setTrailerLine Method

Syntax **setTrailerLine (lineNumber: *int32*, text: *string*, doubleWidth: *boolean*): void { raises-exception, use after open-claim-enable }**

Parameter	Description
<i>lineNumber</i>	Line number of the trailer line to set.
<i>text</i>	Text to which to set the trailer line.
<i>doubleWidth</i>	Print this line in double wide characters.

Remarks Sets one of the fiscal receipt trailer lines. The text set by this method will be stored by the Fiscal Printer and retained across power losses.

The *lineNumber* parameter must be between 1 and the value of the **NumTrailerLines** property. If *text* is an empty string (“”), then the trailer line is unset and will not be printed. The *doubleWidth* characters will be printed if the Fiscal Printer supports them. See the **CapDoubleWidth** property to determine if they are supported. This method is only supported if **CapSetTrailer** is true. This method can only be called while **DayOpened** is false.

Errors A UpoException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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"> • The Fiscal Printer does not support setting the receipt trailer lines (see the CapSetTrailer property), or • The Fiscal Printer has already begun the fiscal day (see the DayOpened property), or • the <i>lineNumber</i> parameter was invalid.
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_BAD_ITEM_DESCRIPTION: The <i>text</i> parameter is too long or contains a reserved word.

See Also **CapDoubleWidth** Property, **CapSetTrailer** Property, **DayOpened** Property, **NumTrailerLines** Property.

setVatTable Method

Syntax	setVatTable (): void { raises-exception, use after open-claim-enable }				
Remarks	Sends the VAT table built inside the Service to the Fiscal Printer. The VAT table is built one entry at a time using the setVatValue method. This method is only supported if CapHasVatTable is true. This method can only be called while DayOpened is false.				
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16. Some possible values of the exception’s <i>ErrorCode</i> property are: <table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>E_ILLEGAL</td> <td>One of the following errors occurred: <ul style="list-style-type: none"> • The Fiscal Printer does not support VAT tables (see the CapHasVatTable property), or • The Fiscal Printer has already begun the fiscal day (see the DayOpened property). </td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	One of the following errors occurred: <ul style="list-style-type: none"> • The Fiscal Printer does not support VAT tables (see the CapHasVatTable property), or • The Fiscal Printer has already begun the fiscal day (see the DayOpened property).
Value	Meaning				
E_ILLEGAL	One of the following errors occurred: <ul style="list-style-type: none"> • The Fiscal Printer does not support VAT tables (see the CapHasVatTable property), or • The Fiscal Printer has already begun the fiscal day (see the DayOpened property). 				
See Also	CapHasVatTable property, DayOpened property, setVatValue Method.				

setVatValue Method

Syntax **setVatValue (vatID: *int32*, vatValue: *string*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>vatID</i>	Index of the VAT table entry to set.
<i>vatValue</i>	Tax value as a percentage.

Remarks Sets the value of a specific VAT class in the VAT table. The VAT table is built one entry at a time in the Service using this method. The entire table is then sent to the Fiscal Printer at one time using the **setVatTable** method.

This method is only supported if **CapHasVatTable** is true. This method can only be called while **DayOpened** is false.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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"> • The Fiscal Printer does not support VAT tables (see the CapHasVatTable property), or • The Fiscal Printer has already begun the fiscal day (see the DayOpened property), or • The Fiscal Printer does not support changing an existing VAT value.

See Also **setVatTable** Method.

verifyItem Method

Syntax `verifyItem (itemName: string, vatID: int32):
 void { raises-exception, use after open-claim-enable }`

Parameter	Description
<i>itemName</i>	Item to be verified.
<i>vatID</i>	VAT identifier of the item.

Remarks Compares *itemName* and its *vatID* with the values stored in the Fiscal Printer.

This method is only supported if **CapHasVatTable** is true. This method can only be called while the Fiscal Printer is in the Item List state.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	The Fiscal Printer does not support VAT tables (see the CapHasVatTable property).
E_EXTENDED	<i>ErrorCodeExtended</i> = EFPTR_WRONG_STATE: The Fiscal Printer is not currently in the Item List state. <i>ErrorCodeExtended</i> = EFPTR_BAD_ITEM_DESCRIPTION: The item name is too long or contains a reserved word. (Only applies if AsyncMode is false.) <i>ErrorCodeExtended</i> = EFPTR_BAD_VAT: The VAT parameter is invalid. (Only applies if AsyncMode is false.)

See Also **CapHasVatTable** property, **setVatTable** Method.

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

See Also "Events" on page 15, **directIO** Method.

ErrorEvent

Updated in Release 1.7

```
<< 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 Fiscal 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 attributes:

Attributes	Type	Description
<i>ErrorCode</i>	<i>int32</i>	Error code causing the error event. See a list of Error Codes on page 16.
<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.

- ErrorLocus* *int32* Location of the error, and is set to EL_OUTPUT indicating that the error occurred while processing asynchronous output.
- 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
EFPTR_COVER_OPEN	The Fiscal Printer cover is open.
EFPTR_JRN_EMPTY	The journal station is out of paper.
EFPTR_REC_EMPTY	The receipt station is out of paper.
EFPTR_SLP_EMPTY	A form is not inserted in the slip station.
EFPTR_WRONG_STATE	The requested method could not be executed in the Fiscal Printer's current state.
EFPTR_TECHNICAL_ASSISTANCE	The Fiscal Printer has encountered a severe error condition. Calling for Fiscal Printer technical assistance is required.
EFPTR_CLOCK_ERROR	The Fiscal Printer's internal clock has failed.
EFPTR_FISCAL_MEMORY_FULL	The Fiscal Printer's fiscal memory has been exhausted.
EFPTR_FISCAL_MEMORY_DISCONNECTED	The Fiscal Printer's fiscal memory has been disconnected.
EFPTR_FISCAL_TOTALS_ERROR	The Grand Total in working memory does not match the one in the EPROM.
EFPTR_BAD_ITEM_QUANTITY	The Quantity parameter is invalid.
EFPTR_BAD_ITEM_AMOUNT	The Amount parameter is invalid.
EFPTR_BAD_ITEM_DESCRIPTION	The Description parameters is either too long, contains illegal characters or contains the reserved word.
EFPTR_RECEIPT_TOTAL_OVERFLOW	The receipt total has overflowed.
EFPTR_BAD_VAT	The Vat parameter is invalid.
EFPTR_BAD_PRICE	The Price parameter is invalid.
EFPTR_NEGATIVE_TOTAL	The Fiscal Printer's computed total or subtotal is less than zero.

EFPTR_MISSING_DEVICES	Some of the other devices which according to the local fiscal legislation are to be connected has been disconnected. In some countries in order to use a fiscal Fiscal Printer a full set of peripheral devices are to be connected to the POS (such as cash drawer and customer display). In case one of these devices is not present sales are not allowed.
EFPTR_BAD_LENGTH	The length of the string to be printed as post or pre line is too long.
EFPTR_MISSING_SET_CURRENCY	The Fiscal Printer is expecting the activation of a new currency.

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 all buffered output data, including all asynchronous output. 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 21, "Device Information Reporting Model" on page 26.

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 21.

StatusUpdateEvent**Updated in Release 1.8**

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

Description Notifies the application that a Fiscal 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	
FPTR_SUE_COVER_OPEN	Fiscal Printer cover is open.	
FPTR_SUE_COVER_OK	Fiscal Printer cover is closed.	
FPTR_SUE_JRN_EMPTY	No journal paper.	
FPTR_SUE_JRN_NEAREMPTY	Journal paper is low.	
FPTR_SUE_JRN_PAPEROK	Journal paper is ready.	
FPTR_SUE_REC_EMPTY	No receipt paper.	
FPTR_SUE_REC_NEAREMPTY	Receipt paper is low.	
FPTR_SUE_REC_PAPEROK	Receipt paper is ready.	
FPTR_SUE_SLP_EMPTY	No slip form is inserted, and no slip form has been detected at the entrance to the slip station. (See “Model” on page 238 for further details on slip properties and events.)	
FPTR_SUE_SLP_NEAREMPTY	Almost at the bottom of the slip form.	
FPTR_SUE_SLP_PAPEROK	Slip form is inserted.	
FPTR_SUE_IDLE	All asynchronous output has finished, either successfully or because output has been cleared. The Fiscal 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 63.

Release 1.8 and later – Specific Cover State Reporting

Starting with Release 1.8, **StatusUpdateEvents** for specific stations' covers are supported. If a Fiscal Printer has only one cover or if it cannot determine/report which covers are open, then only the original FPTR_SUE_COVER_OPEN and FPTR_SUE_COVER_OK events should be fired.

For Fiscal Printers supporting multiple covers, the original events should also be fired for compatibility with current applications. In these cases, the station-specific event should be fired **first**, followed by the original event.

If more than one cover is open, the original FPTR_SUE_COVER_OPEN event should only be fired once after a cover is opened. A FPTR_SUE_COVER_OK event should only be fired after all the covers are closed.

The event's *Status* attribute can contain one of the following additional values to indicate a status change.

Value	Meaning
FPTR_SUE_JRN_COVER_OPEN	Journal station cover is open.
FPTR_SUE_JRN_COVER_OK	Journal station cover is closed.
FPTR_SUE_REC_COVER_OPEN	Receipt station cover is open.
FPTR_SUE_REC_COVER_OK	Receipt station cover is closed.
FPTR_SUE_SLP_COVER_OPEN	Slip station cover is open.
FPTR_SUE_SLP_COVER_OK	Slip station cover is closed.

Remarks Enqueued when a significant status event has occurred.

See Also "Events" on page 15.

Hard Totals

This Chapter defines the Hard Totals device category.

Summary

Properties (UML attributes)

<i>Common</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</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
CapStatisticsReporting:	<i>boolean</i>	{ read-only }	1.8	open
CapUpdateStatistics:	<i>boolean</i>	{ read-only }	1.8	open
CheckHealthText:	<i>string</i>	{ read-only }	1.0	open
Claimed:	<i>boolean</i>	{ read-only }	1.0	open
DataCount:	<i>int32</i>	{ read-only }	1.2	Not Supported
DataEventEnabled:	<i>boolean</i>	{ read-write }	1.0	Not Supported
DeviceEnabled:	<i>boolean</i>	{ read-write }	1.0	open
FreezeEvents:	<i>boolean</i>	{ read-write }	1.0	open
OutputID:	<i>int32</i>	{ read-only }	1.0	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 }	1.0	--
DeviceControlDescription:	<i>string</i>	{ read-only }	1.0	--
DeviceControlVersion:	<i>int32</i>	{ read-only }	1.0	--
DeviceServiceDescription:	<i>string</i>	{ read-only }	1.0	open
DeviceServiceVersion:	<i>int32</i>	{ read-only }	1.0	open
PhysicalDeviceDescription:	<i>string</i>	{ read-only }	1.0	open
PhysicalDeviceName:	<i>string</i>	{ read-only }	1.0	open

Properties (Continued)

<i>Specific</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
CapErrorDetection:	<i>boolean</i>	{ read-only }	1.0	open
CapSingleFile:	<i>boolean</i>	{ read-only }	1.0	open
CapTransactions:	<i>boolean</i>	{ read-only }	1.0	open
FreeData:	<i>int32</i>	{ read-only }	1.0	open & enable
NumberOfFiles:	<i>int32</i>	{ read-only }	1.0	open & enable
TotalsSize:	<i>int32</i>	{ read-only }	1.0	open & enable
TransactionInProgress:	<i>boolean</i>	{ read-only }	1.0	open

Methods (UML operations)**Common**

<i>Name</i>	<i>Version</i>
open (logicalDeviceName: <i>string</i>): void { raises-exception }	1.0
close (): void { raises-exception, use after open }	1.0
claim (timeout: <i>int32</i>): void { raises-exception, use after open }	1.0
release (): void { raises-exception, use after open, claim }	1.0
checkHealth (level: <i>int32</i>): void { raises-exception, use after open, enable } ^a	1.0
clearInput (): void { }	<i>Not supported</i>
clearOutput (): void { }	<i>Not supported</i>
directIO (command: <i>int32</i>, inout data: <i>int32</i>, inout obj: <i>object</i>): void { raises-exception, use after open }	1.0
resetStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
retrieveStatistics (inout statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
updateStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8

Specific

beginTrans (): void { raises-exception, use after open, enable }	1.0
claimFile (hTotalsFile: <i>int32</i>, timeout: <i>int32</i>): void { raises-exception, use after open, enable } ^b	1.0
commitTrans (): void { raises-exception, use after open, enable }	1.0

create (<i>fileName: string</i> , <i>inout hTotalsFile: int32</i> , <i>size: int32</i> , <i>errorDetection: boolean</i>): void { raises-exception, use after open, enable } ^a	1.0
delete (<i>fileName: string</i>): void { raises-exception, use after open, enable } ^b	1.0
find (<i>fileName: string</i> , <i>inout hTotalsFile: int32</i> , <i>inout size: int32</i>): void { raises-exception, use after open, enable } ^a	1.0
findByIndex (<i>index: int32</i> , <i>inout fileName: string</i>): void { raises-exception, use after open, enable } ^a	1.0
read (<i>hTotalsFile: int32</i> , <i>inout data: binary</i> , <i>offset: int32</i> , <i>count: int32</i>): void { raises-exception, use after open, enable } ^b	1.0
recalculateValidationData (<i>hTotalsFile: int32</i>): void { raises-exception, use after open, enable } ^b	1.0
releaseFile (<i>hTotalsFile: int32</i>): void { raises-exception, use after open, enable }	1.0
rename (<i>hTotalsFile: int32</i> , <i>fileName: string</i>): void { raises-exception, use after open, enable } ^b	1.0
rollback (): void { raises-exception, use after open, enable }	1.0
setAll (<i>hTotalsFile: int32</i> , <i>value: byte</i>): void { raises-exception, use after open, enable } ^b	1.0
validateData (<i>hTotalsFile: int32</i>): void { raises-exception, use after open, enable } ^b	1.0
write (<i>hTotalsFile: int32</i> , <i>data: binary</i> , <i>offset: int32</i> , <i>count: int32</i>): void { raises-exception, use after open, enable } ^b	1.0

- a. Also requires that no other application has claimed the hard totals device.
- b. Also requires that no other application has claimed the hard totals device or the file on which this method acts.

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>
upos::events::DataEvent		<i>Not Supported</i>	
upos::events::DirectIOEvent			1.0
EventNumber:	<i>int32</i>	{ read-only }	
Data:	<i>int32</i>	{ read-write }	
Obj:	<i>object</i>	{ read-write }	
upos::events::ErrorEvent		<i>Not Supported</i>	
upos::events::OutputCompleteEvent		<i>Not Supported</i>	
upos::events::StatusUpdateEvent			1.3
Status:	<i>int32</i>	{ read-only }	

General Information

The Hard Totals programmatic name is “HardTotals”.

Capabilities

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

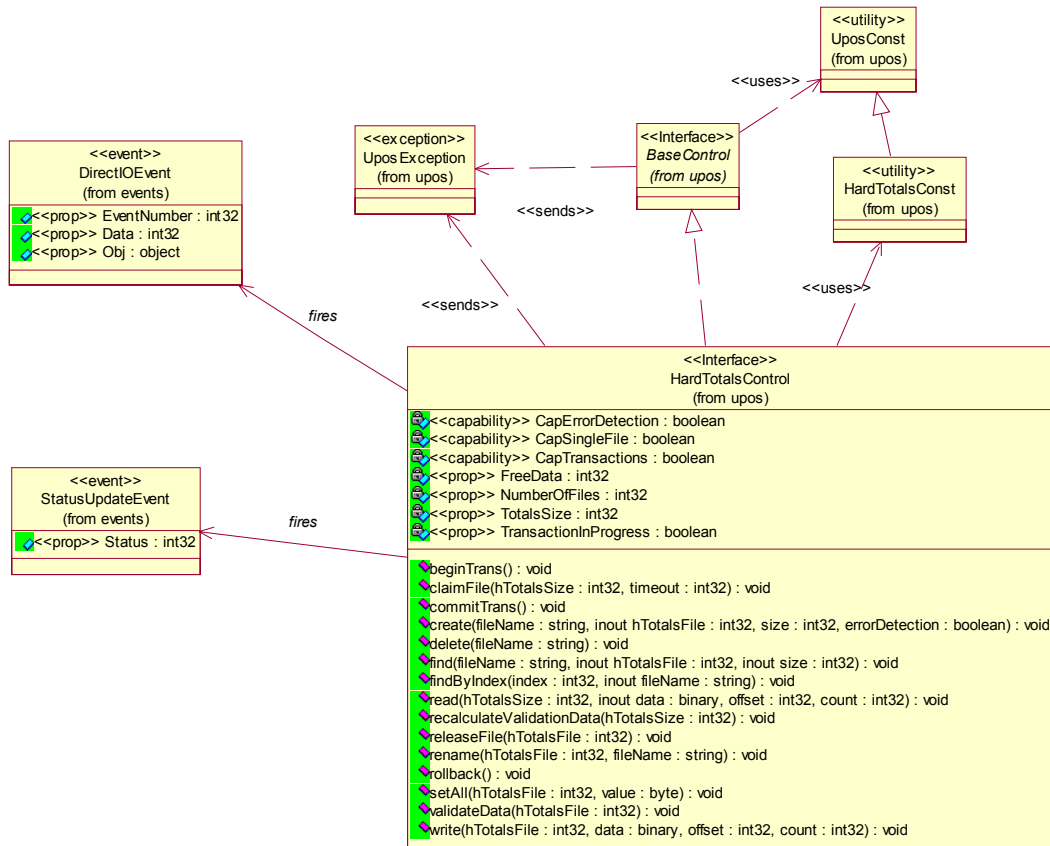
- Supports at least one totals file with the name “” (the empty string) in an area of totals memory. Each totals file is read and written as if it were a sequence of byte data.
- Creates each totals file with a fixed size and may be deleted, initialized, and claimed for exclusive use.

The Hard Totals device may have the following additional capabilities:

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

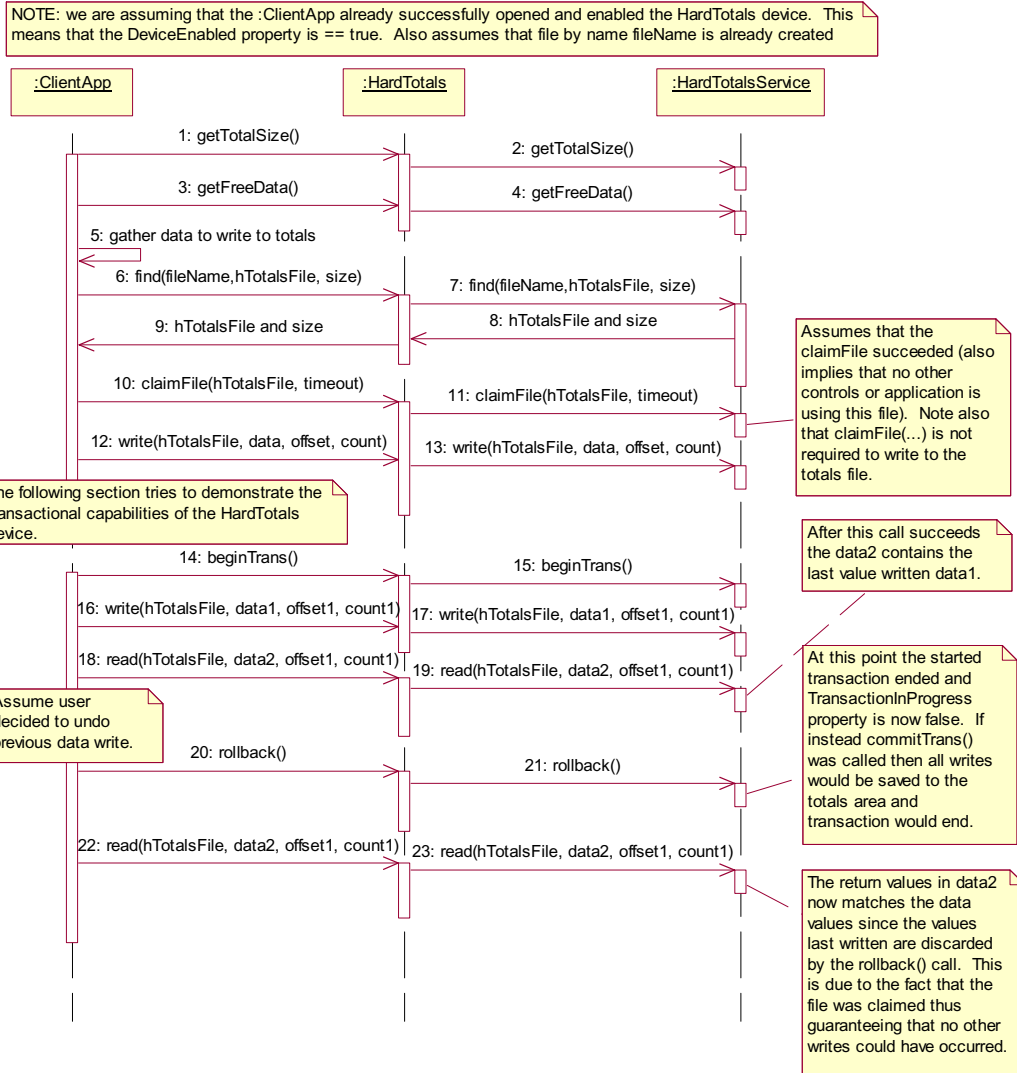
Hard Totals Class Diagram

The following diagram shows the relationships between the Hard Totals classes.



Hard Totals Sequence Diagram *Added in Release 1.7*

The following sequence diagram shows the typical usage of the Hard Totals device, and assumes that a file already exists on the device containing data. It also demonstrates the transactional capabilities of the Hard Totals device.



Model

Totals memory is frequently a limited but secure resource - perhaps of only several thousand bytes of storage. The following is the general model of the Hard Totals:

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

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

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

- The following operations may be performed on a totals file:
 - **read**: Read a series of data bytes.
 - **write**: Write a series of data bytes.
 - **setAll**: Set all the data in a totals file to a value.
 - **find**: Locate an existing totals file by name, and return a file handle and size.
 - **findByIndex**: Enumerate all of the files in the Hard Totals area.
 - **delete**: Delete a totals file by name.
 - **rename**: Rename an existing totals file.
 - **claimFile**: Gain exclusive access to a specific file for use by the claiming application. A timeout value may be specified in case another application maintains access for a period a time.
The common **claim** method may also be used to claim the entire Hard Totals device.
 - **releaseFile**: Release exclusive access to the file.
- The **FreeData** property holds the current number of unassigned data bytes.
- The **TotalsSize** property holds the totals memory size.
- The **NumberOfFiles** property holds the number of totals files that exist in the hard totals device.

- Transaction operations are optionally supported. A transaction is defined as a series of data writes to be applied as an atomic operation to one or more Hard Totals files.

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

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

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

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

The application should **claim** the files used during a transaction so that no other Hard Totals Control claims a file before **commitTrans**, causing the commit to fail, with the exception's *ErrorCode* reflecting an already claimed status.

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

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

When a validation error is reported, it is recommended that the application read all of the data in the totals file. The application will want to determine which portions of data are invalid, and take action based on the results of the reads.
 - **recalculateValidationData** may be called to cause recalculation of all validation data within a totals file. This may be called after recovery has been performed as in the previous paragraph.

- **validateData** may be called to verify that all data within a totals file passes validation.
- Data **writes** automatically cause recalculation of validation data for the validation block or blocks in which the written data resides.
- Since advanced error detection usually imposes a performance penalty, the application may choose to select this feature when each totals file is created.

Device Sharing

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

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

Properties (UML attributes)

CapErrorDetection Property

Syntax	CapErrorDetection: <i>boolean</i> { read-only, access after open }
Remarks	If true, then advanced error detection is supported. 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 16.

CapSingleFile Property

Syntax	CapSingleFile: <i>boolean</i> { read-only, access after open }
Remarks	If true, then only a single file, identified by the empty string (“”), is supported. 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 16.

CapTransactions Property

Syntax	CapTransactions: <i>boolean</i> { read-only, access after open }
Remarks	If true, then transactions are supported. 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 16.

FreeData Property

Syntax	FreeData: <i>int32</i> { read-only, access after open-enable }
Remarks	Holds the number of bytes of unallocated data in the Hard Totals device. It is initialized to an appropriate value when the device is enabled and is updated as files are created and deleted . If creating a file requires some overhead to support the file information, then this overhead is not included in what is reported by this property. This guarantees that a new file of size FreeData may be created. Data writes within a transaction may temporarily reduce what’s reported by this property, since some Hard Totals space may need to be allocated to prepare for the transaction commit. Therefore, the application should ensure that sufficient FreeData is maintained to allow its maximally sized transactions to be performed.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	create Method, write Method.

NumberOfFiles Property

Syntax	NumberOfFiles: <i>int32</i> { read-only, access after open-enable }
Remarks	Holds the number of totals file currently in the Hard Totals device. This property is initialized and kept current while the device is enabled.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	FreeData Property.

TotalsSize Property

Syntax	TotalsSize: <i>int32</i> { read-only, access after open-enable }
Remarks	Holds the size of the Hard Totals area. This size is equal to the largest totals file that can be created if no other files exist. This property is initialized when the device is enabled.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	FreeData Property.

TransactionInProgress Property

Syntax	TransactionInProgress: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the application is within a transaction. 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 16.
See Also	beginTrans Method.

Methods (UML operations)

beginTrans Method

Syntax	beginTrans (): void { raises-exception, use after open-enable }				
Remarks	Marks the beginning of a series of Hard Totals writes that must either be applied as a group or not at all.				
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16. Some possible values of the exception’s <i>ErrorCode</i> property are:				
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>E_ILLEGAL</td> <td>Transactions are not supported by this device.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	Transactions are not supported by this device.
Value	Meaning				
E_ILLEGAL	Transactions are not supported by this device.				
See Also	commitTrans Method, rollback Method.				

claim Method (Common)

Syntax	claim (timeout: <i>int32</i>): void { raises-exception, use after open }						
	The <i>timeout</i> parameter gives the maximum number of milliseconds to wait for exclusive access to be satisfied. If zero, the method attempts to claim the device, then returns the appropriate status immediately. If UPOS_FOREVER (-1), the method waits as long as needed until exclusive access is satisfied.						
Remarks	Requests exclusive access to the device. If any other application has claimed exclusive access to any of the hard totals files by using claimFile , then this claim cannot be satisfied until those files are released by releaseFile . When successful, the Claimed property is changed to true.						
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16. Some possible values of the exception’s <i>ErrorCode</i> property are:						
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>E_ILLEGAL</td> <td>An invalid <i>timeout</i> parameter was specified.</td> </tr> <tr> <td>E_TIMEOUT</td> <td>Another application has exclusive access to the device or one or more of its files and did not relinquish control before <i>timeout</i> milliseconds expired.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	An invalid <i>timeout</i> parameter was specified.	E_TIMEOUT	Another application has exclusive access to the device or one or more of its files and did not relinquish control before <i>timeout</i> milliseconds expired.
Value	Meaning						
E_ILLEGAL	An invalid <i>timeout</i> parameter was specified.						
E_TIMEOUT	Another application has exclusive access to the device or one or more of its files and did not relinquish control before <i>timeout</i> milliseconds expired.						
See Also	“Device Sharing Model” on page 14, release Method, claimFile Method, releaseFile Method.						

claimFile Method**Updated in Release 1.8**

Syntax **claimFile (hTotalsFile: *int32*, timeout: *int32*):**
 void { raises-exception, use after open-enable }

Parameter	Description
<i>hTotalsFile</i>	Handle to the totals file that is to be claimed.
<i>timeout</i>	The time in milliseconds to wait for the file to become available. If zero, the method attempts to claim the file, then returns the appropriate status immediately. If UPOS_FOREVER (-1), the method waits as long as needed until exclusive access is satisfied.

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

If another application has claimed exclusive access to this file by using this method, or if another application has claimed exclusive access to the entire totals area by using **claim**, then this request cannot be satisfied until such claims have been released.

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

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	The handle is invalid, or an invalid <i>timeout</i> parameter was specified.
E_TIMEOUT	The <i>timeout</i> value expired before another application released exclusive access of either the requested totals file or the entire totals area.

See Also **claim** Method, **releaseFile** Method.

commitTrans Method

Syntax **commitTrans ():**
 void { raises-exception, use after open-enable }

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

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	Transactions are not supported by this device, or no transaction is in progress.

See Also **beginTrans** Method, **rollback** Method.

delete Method

Syntax **delete (fileName: *string*):**
 void { raises-exception, use after open-enable }

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

Remarks Deletes the named file.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_CLAIMED	Cannot delete because either the totals file or the entire totals area is claimed by another application.
E_ILLEGAL	The <i>fileName</i> is too long or contains invalid characters.
E_NOEXIST	<i>fileName</i> was not found.

See Also **create Method, find Method, rename Method.**

find Method

Syntax **find (fileName: *string*, inout hTotalsFile: *int32*, inout size: *int32*):**
 void { raises-exception, use after open-enable }

Parameter	Description
<i>fileName</i>	The totals file name to be located.
<i>hTotalsFile</i>	Handle of the totals file. Set by the method.
<i>size</i>	The length of the file in bytes. Set by the method.

Remarks Locates an existing totals file.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_CLAIMED	Cannot find because the entire totals file area is claimed by another application.
E_ILLEGAL	The <i>fileName</i> contains invalid characters.
E_NOEXIST	<i>fileName</i> was not found.

See Also **create Method, delete Method, rename Method.**

findByIndex Method

Syntax **findByIndex (index: *int32*, inout fileName: *string*):
void { raises-exception, use after open-enable }**

Parameter	Description
<i>index</i>	The index of the totals file name to be found.
<i>fileName</i>	The file name associated with <i>index</i> . Set by the method.

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

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

The creation and deletion of files may change the relationship between indices and the file names; the data areas used to manage file names and attributes may be compacted or rearranged as a result. Therefore, the application may need to **claim** the device to ensure that all file names are retrieved successfully.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_CLAIMED	Cannot find because the entire totals file area is claimed by another application.
E_ILLEGAL	The <i>index</i> is greater than the largest file index that is currently defined.

See Also **create** Method, **find** Method.

read Method**Updated in Release 1.7**

Syntax **read (hTotalsFile: *int32*, inout data: *binary*, offset: *int32*, count: *int32*):
 void { raises-exception, use after open-enable }**

Parameter	Description
<i>hTotalsFile</i>	Totals file handle returned from a create or find method.
<i>data</i> ¹	The data buffer in which the totals data will be placed. Array length must be at least <i>count</i> .
<i>offset</i>	Starting offset for the data to be read.
<i>count</i>	Number of bytes of data to read.

Remarks Reads data from a totals file.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_CLAIMED	Cannot read because either the totals file or the entire totals area is claimed by another application.
E_ILLEGAL	The handle is invalid, part of the data range is outside the bounds of the totals file, or <i>data</i> array length is less than <i>count</i> .
E_EXTENDED	<i>ErrorCodeExtended</i> = ETOT_VALIDATION: A validation error has occurred while reading data.

See Also **write Method**

¹. In the OPOS environment, the format of *data* depends upon the value of the **BinaryConversion** property. See **BinaryConversion** property on page A-28.

recalculateValidationData Method

Syntax **recalculateValidationData (hTotalsFile: *int32*):**
 void { raises-exception, use after open-enable }

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

Remarks Recalculates validation data for the specified totals file.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_CLAIMED	Cannot recalculate because either the totals file or the entire totals area is claimed by another application.
E_ILLEGAL	The handle is invalid, or advanced error detection is either not supported by the Service or by this file.

release Method (Common)

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

Remarks Releases exclusive access to the device.

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

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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 14, **claim** Method, **claimFile** Method.

releaseFile Method

Syntax **releaseFile (hTotalsFile: *int32*):**
 void { raises-exception, use after open-enable }

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

Remarks Releases exclusive access to a specific file.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	The handle is invalid, or the specified file is not claimed by this application.

See Also **claim** Method, **claimFile** Method.

rename Method

Syntax **rename (hTotalsFile: *int32*, fileName: *string*):**
 void { raises-exception, use after open-enable }

Parameter	Description
<i>hTotalsFile</i>	The handle of the totals file to be renamed.
<i>fileName</i>	The new name to be assigned to the file. Must be no longer than 10 characters. All displayable ASCII characters (0x20 through 0x7F) are valid.

Remarks Renames a totals file.

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

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_CLAIMED	Cannot rename because either the totals file or the entire totals area is claimed by another application.
E_ILLEGAL	The handle is invalid, the <i>fileName</i> contains invalid characters, or the CapSingleFile property is true.
E_EXISTS	<i>fileName</i> already exists.

See Also **CapSingleFile** Property.

rollback Method

Syntax	rollback (): void { raises-exception, use after open-enable }				
Remarks	Ends the current transaction. All writes between the previous beginTrans and this method are discarded; they are not saved to the Hard Totals areas.				
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16. Some possible values of the exception’s <i>ErrorCode</i> property are:				
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>E_ILLEGAL</td> <td>Transactions are not supported by this device, or no transaction is in progress.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	Transactions are not supported by this device, or no transaction is in progress.
Value	Meaning				
E_ILLEGAL	Transactions are not supported by this device, or no transaction is in progress.				
See Also	beginTrans Method, commitTrans Method.				

setAll Method**Updated in Release 1.7**

Syntax	setAll (hTotalsFile: <i>int32</i>, value: <i>byte</i>): void { raises-exception, use after open-enable }						
	<table> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><i>hTotalsFile</i></td> <td>Handle of a totals file.</td> </tr> <tr> <td><i>value</i></td> <td>Value to set all locations to in totals file.</td> </tr> </tbody> </table>	Parameter	Description	<i>hTotalsFile</i>	Handle of a totals file.	<i>value</i>	Value to set all locations to in totals file.
Parameter	Description						
<i>hTotalsFile</i>	Handle of a totals file.						
<i>value</i>	Value to set all locations to in totals file.						
Remarks	Sets all the data in a totals file to the specified value.						
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16. Some possible values of the exception’s <i>ErrorCode</i> property are:						
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>E_CLAIMED</td> <td>Cannot set because either the totals file or the entire totals area is claimed by another application.</td> </tr> <tr> <td>E_ILLEGAL</td> <td>The handle is invalid.</td> </tr> </tbody> </table>	Value	Meaning	E_CLAIMED	Cannot set because either the totals file or the entire totals area is claimed by another application.	E_ILLEGAL	The handle is invalid.
Value	Meaning						
E_CLAIMED	Cannot set because either the totals file or the entire totals area is claimed by another application.						
E_ILLEGAL	The handle is invalid.						

validateData Method

Syntax **validateData (hTotalsFile: *int32*):
 void { raises-exception, use after open-enable }**

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

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

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_CLAIMED	Cannot validate because either the totals file or the entire totals area is claimed by another application.
E_ILLEGAL	The handle is invalid, or advanced error detection is either not supported by the Service or by this file.

write Method**Updated in Release 1.7**

Syntax **write (hTotalsFile: *int32*, data: *binary*, offset: *int32*, count: *int32*):
 void { raises-exception, use after open-enable }**

Parameter	Description
<i>hTotalsFile</i>	Totals file handle returned from a create or find method.
<i>data</i> ²	Data buffer containing the totals data to be written.
<i>offset</i>	Starting offset for the data to be written.
<i>count</i>	Number of bytes of <i>data</i> to write.

Remarks Writes *data* to a totals file.

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

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_CLAIMED	Cannot write because either the totals file or the entire totals area is claimed by another application.
E_ILLEGAL	The handle is invalid, or part or all of the data range is outside the bounds of the totals file.
E_EXTENDED	<i>ErrorCodeExtended</i> = ETOT_NOROOM: Cannot write because a transaction is in progress, and there is not enough free space to prepare for the transaction commit. <i>ErrorCodeExtended</i> = ETOT_VALIDATION: A validation error has occurred while reading data.

See Also **read** Method, **beginTrans** Method, **commitTrans** Method, **rollback** Method, **FreeData** Property.

². In the OPOS environment, the format of *data* depends upon the value of the **BinaryConversion** property. See **BinaryConversion** property on page A-28.

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

See Also "Events" on page 15, **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 a Hard Totals device.

Attributes This event contains the following attribute:

Attribute	Type	Description
<i>Status</i>	<i>int32</i>	Reports a change in the power state of a Hard Totals device.

Note that Release 1.3 added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See "StatusUpdateEvent" description on page 63.

Remarks Enqueued when the Hard Totals device detects a power state change.

See Also "Events" on page 15.

Keylock

This Chapter defines the Keylock device category.

Summary

Properties (UML attributes)

<i>Common</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</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
CapStatisticsReporting:	<i>boolean</i>	{ read-only }	1.8	open
CapUpdateStatistics:	<i>boolean</i>	{ read-only }	1.8	open
CheckHealthText:	<i>string</i>	{ read-only }	1.0	open
Claimed:	<i>boolean</i>	{ read-only }	1.0	open
DataCount:	<i>int32</i>	{ read-only }	1.2	Not Supported
DataEventEnabled:	<i>boolean</i>	{ read-write }	1.0	Not Supported
DeviceEnabled:	<i>boolean</i>	{ read-write }	1.0	open
FreezeEvents:	<i>boolean</i>	{ read-write }	1.0	open
OutputID:	<i>int32</i>	{ read-only }	1.0	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 }	1.0	--
DeviceControlDescription:	<i>string</i>	{ read-only }	1.0	--
DeviceControlVersion:	<i>int32</i>	{ read-only }	1.0	--
DeviceServiceDescription:	<i>string</i>	{ read-only }	1.0	open
DeviceServiceVersion:	<i>int32</i>	{ read-only }	1.0	open
PhysicalDeviceDescription:	<i>string</i>	{ read-only }	1.0	open
PhysicalDeviceName:	<i>string</i>	{ read-only }	1.0	open
<i>Specific</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
KeyPosition:	<i>int32</i>	{ read-only }	1.0	open & enable
PositionCount:	<i>int32</i>	{ read-only }	1.0	open

Methods (UML operations)**Common**

<i>Name</i>	<i>Version</i>
open (logicalDeviceName: <i>string</i>): void { raises-exception }	1.0
close (): void { raises-exception, use after open }	1.0
claim (timeout: <i>int32</i>): void { raises-exception, use after open }	1.0
release (): void { raises-exception, use after open, claim }	1.0
checkHealth (level: <i>int32</i>): void { raises-exception, use after open, enable }	1.0
clearInput (): void { }	<i>Not supported</i>
clearOutput (): void { }	<i>Not supported</i>
directIO (command: <i>int32</i>, inout data: <i>int32</i>, inout obj: <i>object</i>): void { raises-exception, use after open }	1.0
resetStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
retrieveStatistics (inout statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
updateStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8

Specific

<i>Name</i>	<i>Version</i>
waitForKeylockChange (keyPosition: <i>int32</i>, timeout: <i>int32</i>): void { raises-exception, use after open, enable }	1.0

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>
upos::events::DataEvent		<i>Not Supported</i>	
upos::events::DirectIOEvent			1.0
EventNumber:	<i>int32</i>	{ read-only }	
Data:	<i>int32</i>	{ read-write }	
Obj:	<i>object</i>	{ read-write }	
upos::events::ErrorEvent		<i>Not Supported</i>	
upos::events::OutputCompleteEvent		<i>Not Supported</i>	
upos::events::StatusUpdateEvent			1.0
Status:	<i>int32</i>	{ read-only }	

General Information

The Keylock programmatic name is “Keylock”.

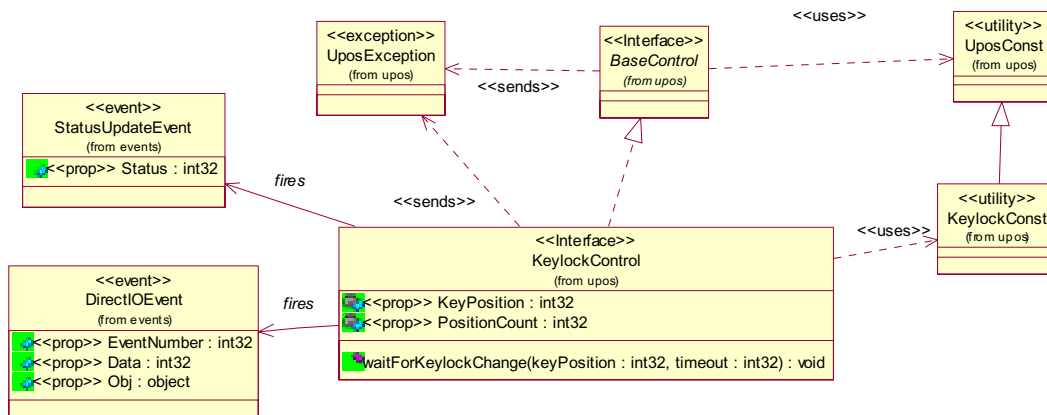
Capabilities

The keylock has the following minimal set of capabilities:

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

Keylock Class Diagram *Updated in Release 1.7*

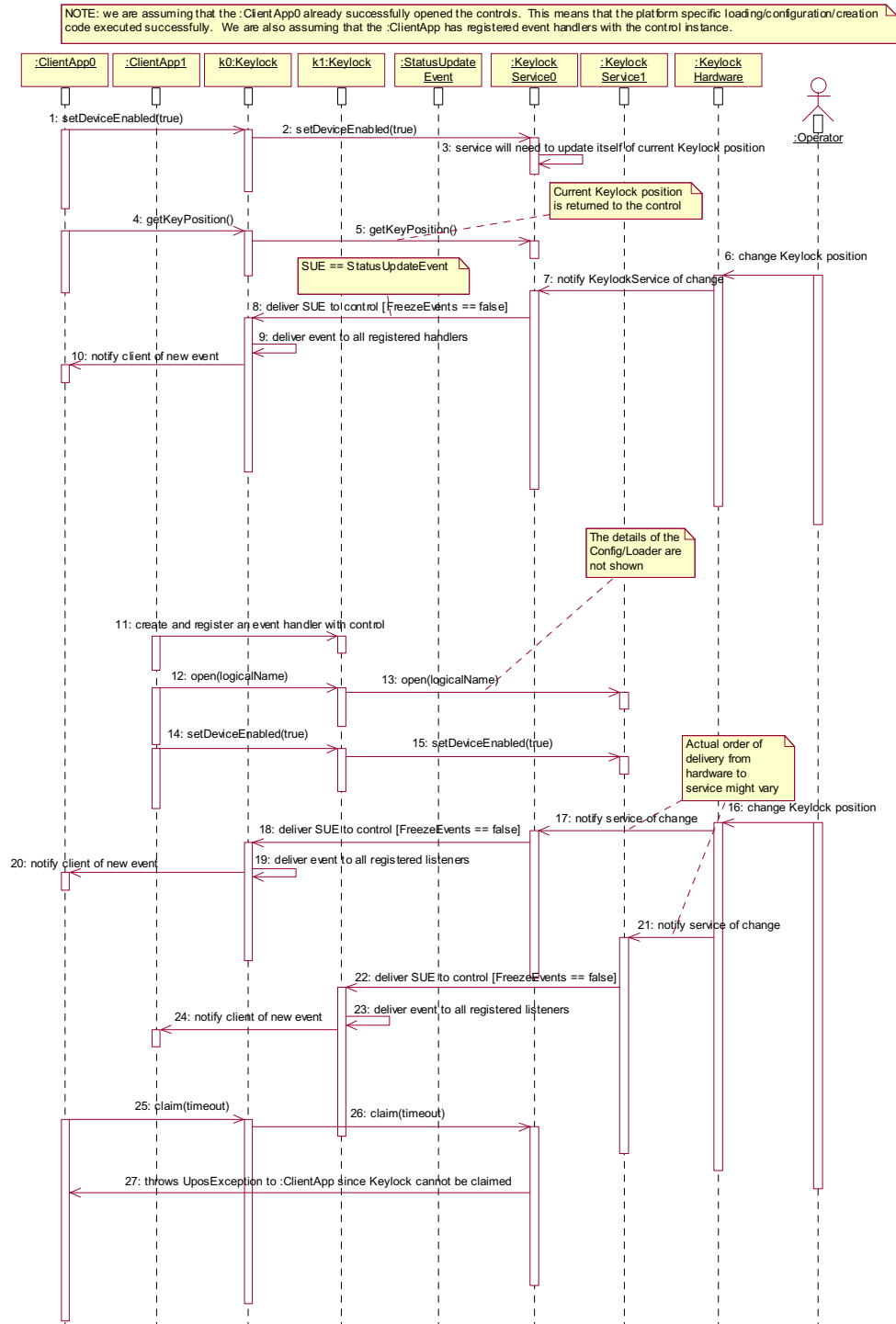
The following diagram shows the relationships between the Keylock classes.



Keylock Sequence Diagram

Added in Release 1.7

The following sequence diagram show the typical usage of the Keylock; as well as showing the unique sharing model of the Keylock. This is the only device that is a purely shareable device.



Model

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

- LOCK_KP_LOCK 1
- LOCK_KP_NORM 2
- LOCK_KP_SUPR 3

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

Device Sharing

The keylock is a sharable device. Its device sharing rules are:

- After opening and enabling the device, the application may access all properties and methods and will receive status update events.
- If more than one application has opened and enabled the device, each of these applications may access its properties and methods. Status update events are fired to all of these applications.
- The keylock may not be claimed for exclusive access. Therefore, if an application calls **claim** or **release**, these methods will always raise a `UposException`.
- See the “Summary” table for precise usage prerequisites.

Properties (UML attributes)

KeyPosition Property

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

Remarks Holds a value that indicates the keylock position.

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

This property has one of the following values:

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

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 16.

See Also **PositionCount** Property, **StatusUpdateEvent**.

PositionCount Property

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

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

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

Errors A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

Methods (UML operations)

waitForKeylockChange Method

Syntax `waitForKeylockChange (keyPosition: int32, timeout: int32):
 void { raises-exception, use after open-enable }`

Parameter	Description
<i>keyPosition</i>	Requested keylock position. See values below.
<i>timeout</i>	Maximum number of milliseconds to wait for the keylock before returning control back to the application. If zero, the method then returns immediately. If UPOS_FOREVER (-1), the method waits as long as needed until the requested key position is satisfied or an error occurs.

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

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

Remarks Waits for a specified keylock position to be set.
If the keylock position specified by the *keyPosition* parameter is the same as the current keylock position, then the method returns immediately.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	An invalid parameter value was specified.
E_TIMEOUT	The <i>timeout</i> period expired before the requested keylock positioning occurred.

See Also **PositionCount** 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 Keylock 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 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 Keylock devices which may not have any knowledge of the Service's need for this event.

See Also "Events" on page 15, **directIO** Method.

StatusUpdateEvent

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

Description Notifies the application when the keylock position changes.

Attributes This event contains the following attribute:

Attribute	Type	Description
<i>Status</i>	<i>int32</i>	The key position in the Keylock.

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

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

Note that Release 1.3 added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See “StatusUpdateEvent” description on page 63.

Remarks This event is enqueued when a keylock switch position undergoes a change or if Power State Reporting is enabled and a change in the power state is detected.

See Also **PositionCount** Property, “Events” on page 15.

Line Display

This Chapter defines the Line Display device category.

Summary

Properties (UML attributes)

<i>Common</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</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
CapStatisticsReporting:	<i>boolean</i>	{ read-only }	1.8	open
CapUpdateStatistics:	<i>boolean</i>	{ read-only }	1.8	open
CheckHealthText:	<i>string</i>	{ read-only }	1.0	open
Claimed:	<i>boolean</i>	{ read-only }	1.0	open
DataCount:	<i>int32</i>	{ read-only }	1.2	Not Supported
DataEventEnabled:	<i>boolean</i>	{ read-write }	1.0	Not Supported
DeviceEnabled:	<i>boolean</i>	{ read-write }	1.0	open & claim
FreezeEvents:	<i>boolean</i>	{ read-write }	1.0	open
OutputID:	<i>int32</i>	{ read-only }	1.0	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 }	1.0	--
DeviceControlDescription:	<i>string</i>	{ read-only }	1.0	--
DeviceControlVersion:	<i>int32</i>	{ read-only }	1.0	--
DeviceServiceDescription:	<i>string</i>	{ read-only }	1.0	open
DeviceServiceVersion:	<i>int32</i>	{ read-only }	1.0	open
PhysicalDeviceDescription:	<i>string</i>	{ read-only }	1.0	open
PhysicalDeviceName:	<i>string</i>	{ read-only }	1.0	open

Properties (Continued)

<i>Specific</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
CapBlink:	<i>int32</i>	{ read-only }	1.0	open
CapBitmap:	<i>boolean</i>	{ read-only }	1.7	open
CapBlinkRate:	<i>boolean</i>	{ read-only }	1.6	open
CapBrightness:	<i>boolean</i>	{ read-only }	1.0	open
CapCharacterSet:	<i>int32</i>	{ read-only }	1.0	open
CapCursorType:	<i>int32</i>	{ read-only }	1.6	open
CapCustomGlyph:	<i>boolean</i>	{ read-only }	1.6	open
CapDescriptors:	<i>boolean</i>	{ read-only }	1.0	open
CapHMarquee:	<i>boolean</i>	{ read-only }	1.0	open
CapICharWait:	<i>boolean</i>	{ read-only }	1.0	open
CapMapCharacterSet:	<i>boolean</i>	{ read-only }	1.7	open
CapReadBack:	<i>int32</i>	{ read-only }	1.6	open
CapReverse:	<i>int32</i>	{ read-only }	1.6	open
CapScreenMode:	<i>boolean</i>	{ read-only }	1.7	open
CapVMarquee:	<i>boolean</i>	{ read-only }	1.0	open
BlinkRate:	<i>int32</i>	{ read-write }	1.6	open
CharacterSet:	<i>int32</i>	{ read-write }	1.0	open, claim, & enable
CharacterSetList:	<i>string</i>	{ read-only }	1.0	open
Columns:	<i>int32</i>	{ read-only }	1.0	open
CurrentWindow:	<i>int32</i>	{ read-write }	1.0	open
CursorColumn:	<i>int32</i>	{ read-write }	1.0	open
CursorRow:	<i>int32</i>	{ read-write }	1.0	open
CursorType:	<i>int32</i>	{ read-write }	1.6	open
CursorUpdate:	<i>boolean</i>	{ read-write }	1.0	open
CustomGlyphList:	<i>string</i>	{ read-only }	1.6	open
DeviceBrightness:	<i>int32</i>	{ read-write }	1.0	open, claim, & enable
DeviceColumns:	<i>int32</i>	{ read-only }	1.0	open
DeviceDescriptors:	<i>int32</i>	{ read-only }	1.0	open
DeviceRows:	<i>int32</i>	{ read-only }	1.0	open
DeviceWindows:	<i>int32</i>	{ read-only }	1.0	open
GlyphHeight:	<i>int32</i>	{ read-only }	1.6	open
GlyphWidth:	<i>int32</i>	{ read-only }	1.6	open
InterCharacterWait:	<i>int32</i>	{ read-write }	1.0	open
MapCharacterSet:	<i>boolean</i>	{ read-write }	1.7	open
MarqueeFormat:	<i>int32</i>	{ read-write }	1.0	open
MarqueeRepeatWait:	<i>int32</i>	{ read-write }	1.0	open

Properties (Continued)

<i>Specific</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
MarqueeType:	<i>int32</i>	{ read-write }	1.0	open
MarqueeUnitWait:	<i>int32</i>	{ read-write }	1.0	open
MaximumX:	<i>int32</i>	{ read-only }	1.7	open
MaximumY:	<i>int32</i>	{ read-only }	1.7	open
Rows:	<i>int32</i>	{ read-only }	1.0	open
ScreenMode:	<i>int32</i>	{ read-write }	1.7	open & claim
ScreenModeList:	<i>string</i>	{ read-only }	1.7	open

Methods (UML operations)**Common**

<i>Name</i>	<i>Version</i>
open (logicalDeviceName: <i>string</i>): void { raises-exception }	1.0
close (): void { raises-exception, use after open }	1.0
claim (timeout: <i>int32</i>): void { raises-exception, use after open }	1.0
release (): void { raises-exception, use after open, claim }	1.0
checkHealth (level: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.0
clearInput (): void { raises-exception, use after open, claim }	<i>Not supported</i>
clearOutput (): void { raises-exception, use after open, claim }	<i>Not supported</i>
directIO (command: <i>int32</i>, inout data: <i>int32</i>, inout obj: <i>object</i>): void { raises-exception, use after open }	1.0
resetStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
retrieveStatistics (inout statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
updateStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8

Specific

<i>Name</i>	<i>Version</i>
clearText (): void { raises-exception, use after open, claim, enable }	1.0
displayText (data: <i>string</i>, attribute: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.0

displayTextAt (row: <i>int32</i> , column: <i>int32</i> , data: <i>string</i> , attribute: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.0
scrollText (direction: <i>int32</i> , units: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.0
clearDescriptors (): void { raises-exception, use after open, claim, enable }	1.0
setDescriptor (descriptor: <i>int32</i> , attribute: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.0
createWindow (viewportRow: <i>int32</i> , viewportColumn: <i>int32</i> , viewportHeight: <i>int32</i> , viewportWidth: <i>int32</i> , windowHeight: <i>int32</i> , windowWidth: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.0
destroyWindow (): void { raises-exception, use after open, claim, enable }	1.0
refreshWindow (window: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.0
defineGlyph (glyphCode: <i>int32</i> , glyph: <i>binary</i>): void { raises-exception, use after open, claim, enable }	1.6
readCharacterAtCursor (inout cursorData: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.6
displayBitmap (fileName: <i>string</i> , width: <i>int32</i> , alignmentX: <i>int32</i> , alignmentY: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.7
setBitmap (bitmapNumber: <i>int32</i> , fileName: <i>string</i> , width: <i>int32</i> , alignmentX: <i>int32</i> , alignmentY: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.7

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>
upos::events::DataEvent		<i>Not Supported</i>	
upos::events::DirectIOEvent			1.0
EventNumber:	<i>int32</i>	{ read-only }	
Data:	<i>int32</i>	{ read-write }	
Obj:	<i>object</i>	{ read-write }	
upos::events::ErrorEvent		<i>Not Supported</i>	
upos::events::OutputCompleteEvent		<i>Not Supported</i>	
upos::events::StatusUpdateEvent			1.3
Status:	<i>int32</i>	{ read-only }	

General Information

The Line Display programmatic name is “LineDisplay”.

Capabilities

Updated in Version 1.7

The Line Display has the following capability:

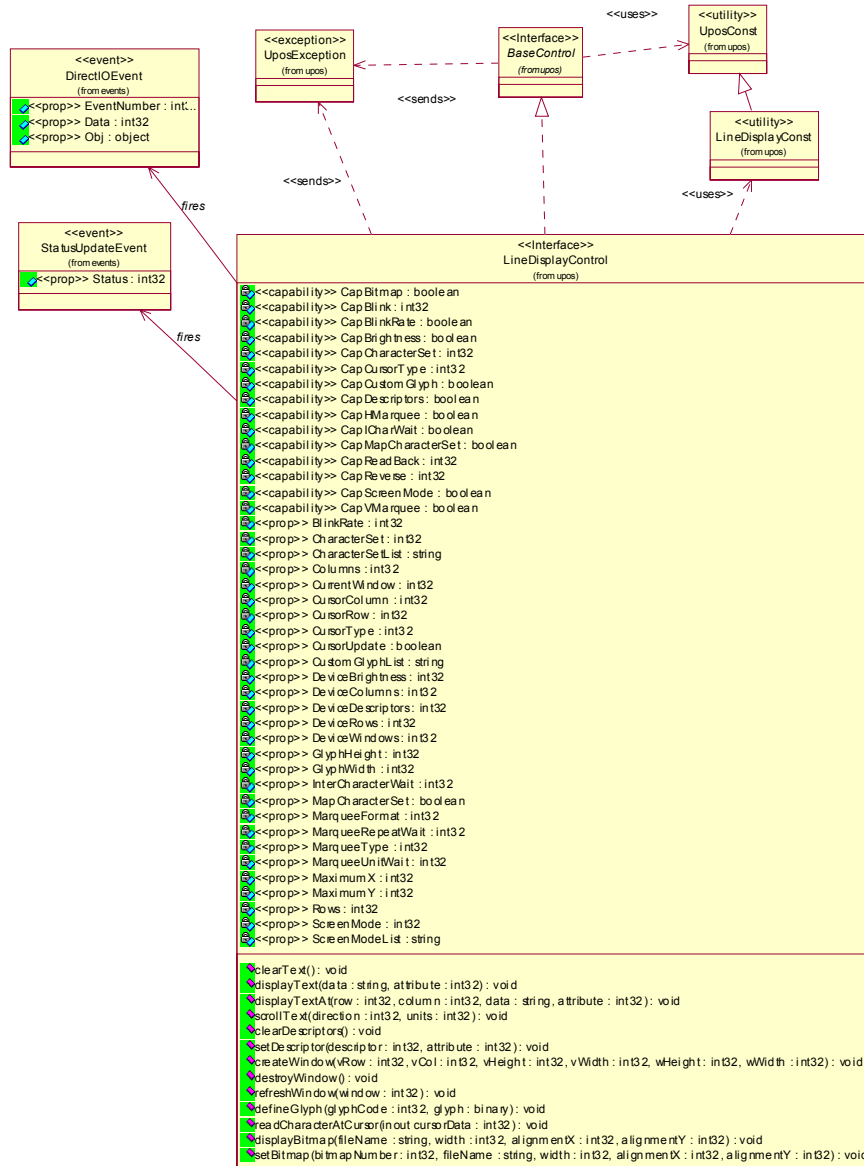
- Supports text character display. The default mode (or perhaps only mode) of the display is character display output.

The line display may also have the following additional capabilities:

- Supports windowing with marquee-like scrolling of the window. The display may support vertical or horizontal marquees, or both.
- Supports a waiting period between displaying characters, for a teletype effect.
- Supports character-level or device-level blinking at adjustable blink rates.
- Supports character-level or device-level reverse video.
- Supports one or more descriptors. Descriptors are small indicators with a fixed label, and are typically used to indicate transaction states such as item, total, and change.
- Supports device brightness control, with one or more levels of device dimming. All devices support brightness levels of “normal” and “blank” (at least through software support), but some devices also support one or more levels of dimming.
- Supports various cursor attributes including underline, block, and reverse video.
- Supports “glyphs” which represent pixel level user definition of character cells.
- Supports changing screen modes - the number of rows and columns supported by the device.
- Supports setting and displaying bitmaps. Can also support the addressing of individual pixels or dots using this functionality.

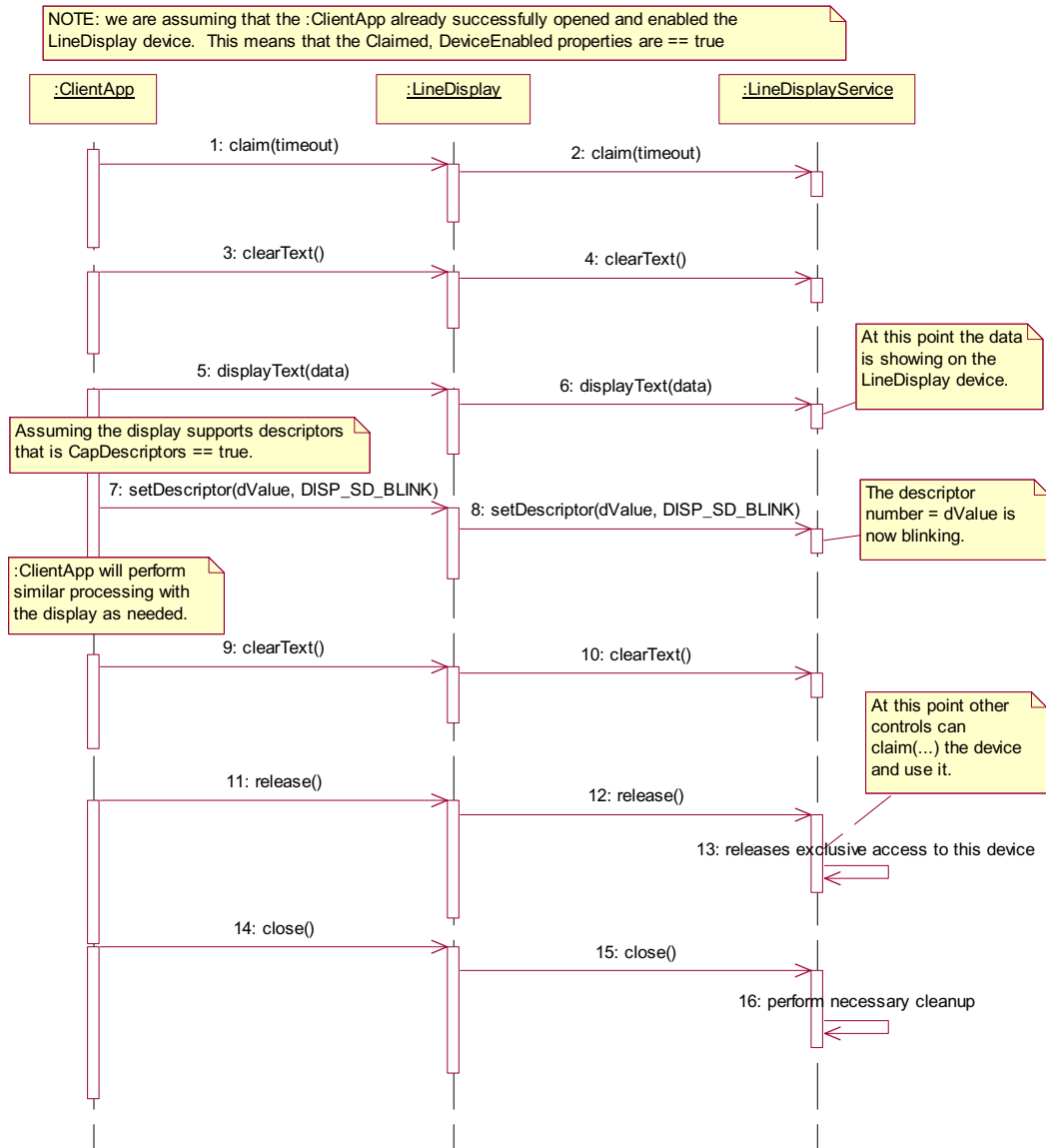
Line Display Class Diagram *Updated in Release 1.7*

The following diagram shows the relationships between the Line Display classes.



Line Display Sequence Diagram *Added in Release 1.7*

The following sequence diagram shows the typical usage of the Line Display device.



Model

Updated in Release 1.7

The general model of a line display consists of:

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

Display Modes

- **Immediate Mode**
In effect when **MarqueeType** is DISP_MT_NONE and **InterCharacterWait** is zero.
If the window is bigger than the viewport, then only those characters which map into the viewport will be seen.
- **Teletype Mode**
In effect when **MarqueeType** is DISP_MT_NONE and **InterCharacterWait** is not zero.
Calls to **displayText** and **displayTextAt** are enqueued and processed in the order they are received. **InterCharacterWait** specifies the time to wait between outputting each character. **InterCharacterWait** only applies to those characters within the viewport.
- **Marquee Mode**
In effect when **MarqueeType** is not DISP_MT_NONE.
The window must be bigger than the viewport.

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

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

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

It is illegal to use **displayText**, **displayTextAt**, **clearText**, **refreshWindow**, and **scrollText** unless in **Marquee Init Mode** or **Marquee Off Mode**.

Data Characters and Escape Sequences

Added in Release 1.7

The default character set of all line displays is assumed to support at least the ASCII characters 0x20 through 0x7F, which include spaces, digits, uppercase, lowercase, and some special characters. If the line display does not support lowercase characters, then the Service may translate them to uppercase.

Starting with Release 1.7, escape sequences are supported.

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.

The following escape sequences are recognized within the string data of the **displayText** and **displayTextAt** methods. If an escape sequence specifies an operation that is not supported by the line display, then it is ignored.

Commands Perform the indicated action.

Name	Data	Remarks
Display bitmap	ESC #B	Displays the pre-stored bitmap. The character '#' is replaced by the bitmap number. See setBitmap method. (If this bitmap is not defined, or if the bitmap cannot be properly displayed, then the escape sequence is ignored.)

Characteristics These are reset at the end of each display method or by a “Normal” sequence.

Name	Data	Remarks
Reverse video	ESC rvC	Displays in reverse video format.
Blink	ESC kC	Displays as blinking characters.
Normal	ESC N	Restores line display characteristics to normal condition.

Device Sharing

The line display is an exclusive-use device, as follows:

- The application must claim the device before enabling it.
- The application must claim and enable the device before accessing some properties or calling methods that update the device.
- See the “Summary” table for precise usage prerequisites.

Properties (UML attributes)

BlinkRate Property

Added in Release 1.6

Syntax	BlinkRate: <i>int32</i> { read-write, access after open }				
Remarks	<p>Contains the blink cycle time in milliseconds. A blink cycle is the period of time when text completes an on-off-on cycle during blinking. After this property is set, the service will set the blink rate to the closest supported rate and change this property to reflect the actual rate. Performing this approximation is necessary because blink cycles are hardware dependent and probably not controllable at precise millisecond granularity.</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 16.</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>CapBlinkRate is false.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	CapBlinkRate is false.
Value	Meaning				
E_ILLEGAL	CapBlinkRate is false.				
See Also	CapBlinkRate Property.				

CapBitmap Property

Added in Release 1.7

Syntax	CapBitmap: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, then the display of bitmaps is supported.</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 16.

CapBlink Property

Syntax	CapBlink: <i>int32</i> { read-only, access after open }								
Remarks	Holds the character blink capability of the device. It has one of the following values:								
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>DISP_CB_NOBLINK</td> <td>Blinking is not supported. Value is 0.</td> </tr> <tr> <td>DISP_CB_BLINKALL</td> <td>Blinking is supported. The entire contents of the display are either blinking or in a steady state.</td> </tr> <tr> <td>DISP_CB_BLINKEACH</td> <td>Blinking is supported. Each character may be individually set to blink or to be in a steady state.</td> </tr> </tbody> </table>	Value	Meaning	DISP_CB_NOBLINK	Blinking is not supported. Value is 0.	DISP_CB_BLINKALL	Blinking is supported. The entire contents of the display are either blinking or in a steady state.	DISP_CB_BLINKEACH	Blinking is supported. Each character may be individually set to blink or to be in a steady state.
Value	Meaning								
DISP_CB_NOBLINK	Blinking is not supported. Value is 0.								
DISP_CB_BLINKALL	Blinking is supported. The entire contents of the display are either blinking or in a steady state.								
DISP_CB_BLINKEACH	Blinking is supported. Each character may be individually set to blink or to be in a steady state.								
	This property is initialized by the open method.								
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.								

CapBlinkRate Property**Added in Release 1.6**

Syntax	CapBlinkRate: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the device’s blink rate can be controlled and the BlinkRate property is used to indicate the rate at which the display blinks.
	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 16.

CapBrightness Property

Syntax	CapBrightness: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the brightness control is supported.
	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 16.

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> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>DISP_CCS_NUMERIC</td> <td>The default character set supports numeric data, plus space, minus, and period.</td> </tr> <tr> <td>DISP_CCS_ALPHA</td> <td>The default character set supports uppercase alphabetic plus numeric, space, minus, and period.</td> </tr> <tr> <td>DISP_CCS_ASCII</td> <td>The default character set supports all ASCII characters 0x20 through 0x7F.</td> </tr> <tr> <td>DISP_CCS_KANA</td> <td>The default character set supports partial code page 932, including ASCII characters 0x20 through 0x7F and the Japanese Kana characters 0xA1 through 0xDF, but excluding the Japanese Kanji characters.</td> </tr> <tr> <td>DISP_CCS_KANJI</td> <td>The default character set supports code page 932, including the Shift-JIS Kanji characters, Levels 1 and 2.</td> </tr> <tr> <td>DISP_CCS_UNICODE</td> <td>The default character set supports UNICODE.</td> </tr> </tbody> </table> <p>The default character set may contain a superset of these ranges. The initial CharacterSet property may be examined for additional information.</p> <p>This property is initialized by the open method.</p>	Value	Meaning	DISP_CCS_NUMERIC	The default character set supports numeric data, plus space, minus, and period.	DISP_CCS_ALPHA	The default character set supports uppercase alphabetic plus numeric, space, minus, and period.	DISP_CCS_ASCII	The default character set supports all ASCII characters 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_NUMERIC	The default character set supports numeric data, plus space, minus, and period.														
DISP_CCS_ALPHA	The default character set supports uppercase alphabetic plus numeric, space, minus, and period.														
DISP_CCS_ASCII	The default character set supports all ASCII characters 0x20 through 0x7F.														
DISP_CCS_KANA	The default character set supports partial code page 932, including ASCII characters 0x20 through 0x7F and the Japanese Kana characters 0xA1 through 0xDF, but excluding the Japanese Kanji characters.														
DISP_CCS_KANJI	The default character set supports code page 932, including the Shift-JIS Kanji characters, Levels 1 and 2.														
DISP_CCS_UNICODE	The default character set supports UNICODE.														
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.														
See Also	CharacterSet Property.														

CapCursorType Property**Updated in Release 1.8**

Syntax	CapCursorType: <i>int32</i> { read-only, access after open }																		
Remarks	<p>Holds a bitwise indication of the cursor types supported by the device and selectable via the CursorType property. The following are the values:</p> <table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>DISP_CCT_NONE</td> <td>Cursor is not displayable.</td> </tr> <tr> <td>DISP_CCT_FIXED</td> <td>Cursor is always displayed.</td> </tr> <tr> <td>DISP_CCT_BLOCK</td> <td>Cursor is displayable as a block.</td> </tr> <tr> <td>DISP_CCT_HALFBLOCK</td> <td>Cursor is displayable as a halfblock.</td> </tr> <tr> <td>DISP_CCT_UNDERLINE</td> <td>Cursor is displayable as an underline.</td> </tr> <tr> <td>DISP_CCT_REVERSE</td> <td>Cursor is displayable in reverse video.</td> </tr> <tr> <td>DISP_CCT_BLINK</td> <td>A blinking cursor is supported.</td> </tr> <tr> <td>DISP_CCT_OTHER</td> <td>Cursor is displayable but form is unknown.</td> </tr> </tbody> </table> <p>If DISP_CCT_NONE is set, then none of the other values will be set. This is because the cursor is not displayable.</p> <p>If DISP_CCT_FIXED is set, DISP_CCT_BLINK may be set, and one and only one of the other values will also be set. This other value will indicate the cursor type that is always displayed.</p> <p>This property is initialized by the open method.</p>	Value	Meaning	DISP_CCT_NONE	Cursor is not displayable.	DISP_CCT_FIXED	Cursor is always displayed.	DISP_CCT_BLOCK	Cursor is displayable as a block.	DISP_CCT_HALFBLOCK	Cursor is displayable as a halfblock.	DISP_CCT_UNDERLINE	Cursor is displayable as an underline.	DISP_CCT_REVERSE	Cursor is displayable in reverse video.	DISP_CCT_BLINK	A blinking cursor is supported.	DISP_CCT_OTHER	Cursor is displayable but form is unknown.
Value	Meaning																		
DISP_CCT_NONE	Cursor is not displayable.																		
DISP_CCT_FIXED	Cursor is always displayed.																		
DISP_CCT_BLOCK	Cursor is displayable as a block.																		
DISP_CCT_HALFBLOCK	Cursor is displayable as a halfblock.																		
DISP_CCT_UNDERLINE	Cursor is displayable as an underline.																		
DISP_CCT_REVERSE	Cursor is displayable in reverse video.																		
DISP_CCT_BLINK	A blinking cursor is supported.																		
DISP_CCT_OTHER	Cursor is displayable but form is unknown.																		
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.																		

CapCustomGlyph Property***Added in Release 1.6***

Syntax	CapCustomGlyph: <i>boolean</i> { read-only, access after open }
Remarks	Holds the glyph definition capability of the device. If true, then the device allows custom glyphs to be defined. 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 16.

CapDescriptors Property

Syntax	CapDescriptors: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the display supports descriptors. 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 16.

CapHMarquee Property

Syntax	CapHMarquee: <i>boolean</i> { read-only, access after open }
Remarks	If true, the display supports horizontal marquee windows. 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 16.

CapICharWait Property

Syntax	CapICharWait: <i>boolean</i> { read-only, access after open }
Remarks	If true, the display supports intercharacter wait. 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 16.

CapMapCharacterSet Property***Added in Release 1.7***

Syntax	CapMapCharacterSet: <i>boolean</i> { read-only, access after open }
Remarks	Defines the ability of the Service to map the characters of the application to the selected character set when displaying data. If CapMapCharacterSet is true, then the Service is able to map the characters to the character sets defined in CharacterSetList . 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 16.
See Also	CharacterSet Property, MapCharacterSet Property, CharacterSetList Property.

CapReadBack Property**Added in Release 1.6**

Syntax	CapReadBack: <i>int32</i> { read-only, access after open }						
Remarks	Holds the capability of the video device to read back the data displayed upon it. It may be one of the following: <table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>DISP_CRB_NONE</td> <td>Read back is not supported.</td> </tr> <tr> <td>DISP_CRB_SINGLE</td> <td>Read back of a single character at a time is supported.</td> </tr> </tbody> </table> <p>This property is initialized by the open method.</p>	Value	Meaning	DISP_CRB_NONE	Read back is not supported.	DISP_CRB_SINGLE	Read back of a single character at a time is supported.
Value	Meaning						
DISP_CRB_NONE	Read back is not supported.						
DISP_CRB_SINGLE	Read back of a single character at a time is supported.						
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.						

CapReverse Property**Added in Release 1.6**

Syntax	CapReverse: <i>int32</i> { read-only, access after open }								
Remarks	Holds the reverse video capability of the device. It may be one of the following: <table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>DISP_CR_NONE</td> <td>Reverse video is not supported. Value is 0.</td> </tr> <tr> <td>DISP_CR_REVERSEALL</td> <td>Reverse video is supported. The entire contents of the display are either in reverse video or normal.</td> </tr> <tr> <td>DISP_CR_REVERSEEACH</td> <td>Reverse video is supported. Each character may be individually set to reverse video or normal.</td> </tr> </tbody> </table> <p>This property is initialized by the open method.</p>	Value	Meaning	DISP_CR_NONE	Reverse video is not supported. Value is 0.	DISP_CR_REVERSEALL	Reverse video is supported. The entire contents of the display are either in reverse video or normal.	DISP_CR_REVERSEEACH	Reverse video is supported. Each character may be individually set to reverse video or normal.
Value	Meaning								
DISP_CR_NONE	Reverse video is not supported. Value is 0.								
DISP_CR_REVERSEALL	Reverse video is supported. The entire contents of the display are either in reverse video or normal.								
DISP_CR_REVERSEEACH	Reverse video is supported. Each character may be individually set to reverse video or normal.								
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.								

CapScreenMode Property**Added in Release 1.7**

Syntax	CapScreenMode: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the display supports changing the screen mode (i.e., the number of text rows and columns on the device). <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 16.
See Also	ScreenMode Property, ScreenModeList Property.

CapVMarquee Property

Syntax	CapVMarquee: <i>boolean</i> { read-only, access after open }
Remarks	If true, the display supports vertical marquee windows. 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 16.

CharacterSet Property

Updated in Release 1.5

Syntax	CharacterSet: <i>int32</i> { read-write, access after open-claim-enable }												
Remarks	Holds the character set for displaying characters. It has one of the following values: <table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>Range 101 - 199</td> <td>Device-specific character sets that do not match a code page or the ASCII or ANSI character sets.</td> </tr> <tr> <td>Range 400 - 990</td> <td>Code page; matches one of the standard values.</td> </tr> <tr> <td>DISP_CS_UNICODE</td> <td>The character set supports UNICODE. The value of this constant is 997.</td> </tr> <tr> <td>DISP_CS_ASCII</td> <td>The ASCII character set, supporting the ASCII characters 0x20 through 0x7F. The value of this constant is 998.</td> </tr> <tr> <td>DISP_CS_ANSI</td> <td>The ANSI character set. The value of this constant is 999.</td> </tr> </tbody> </table> <p>This property is initialized to an appropriate value when the device is first enabled following the open method. This value is guaranteed to support at least the set of characters specified by CapCharacterSet.</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.	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.
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.												
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.												
See Also	CharacterSetList Property, CapCharacterSet Property.												

CharacterSetList Property

Syntax	CharacterSetList: <i>string</i> { read-only, access after open }
Remarks	Holds the character set numbers supported. It consists of ASCII numeric set numbers separated by commas. For example, if the string is “101,850,999”, then the device supports a device-specific character set, code page 850, and the ANSI character set. 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 16.
See Also	CharacterSet Property.

Columns Property

Syntax	Columns: <i>int32</i> { read-only, access after open }
Remarks	<p>Holds the number of columns for this window.</p> <p>For window 0, this property is the same as DeviceColumns. For other windows, it may be less or greater than DeviceColumns.</p> <p>This property is initialized to DeviceColumns by the open method, and is updated when CurrentWindow is set and when createWindow or destroyWindow are called.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	Rows Property.

CurrentWindow Property

Updated in Release 1.6

Syntax	CurrentWindow: <i>int32</i> { read-write, access after open }				
Remarks	<p>Holds the current window to which text is displayed.</p> <p>Several properties are associated with each window: Rows, Columns, CursorRow, CursorColumn, CursorUpdate, CursorType, MarqueeFormat, MarqueeType, MarqueeUnitWait, MarqueeRepeatWait, and InterCharacterWait.</p> <p>When set, this property changes the current window and sets the associated properties to their values for this window.</p> <p>Setting a window does not refresh its viewport. If this window and another window’s viewports overlap, and the other window has changed the viewport, then refreshWindow may be called to restore this window’s viewport contents.</p> <p>This property is initialized to zero – the device window – by the open method, and is updated when createWindow or destroyWindow are called.</p>				
Errors	<p>A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.</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>The new current window value is invalid.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	The new current window value is invalid.
Value	Meaning				
E_ILLEGAL	The new current window value is invalid.				

CursorColumn Property

- Syntax** **CursorColumn: *int32* { read-write, access after open }**
- Remarks** Holds the column in the current window to which the next displayed character will be output.
- Legal values range from zero through **Columns**. (See **displayText** for a note on the interpretation of **CursorColumn = Columns**.)
- This property is initialized to zero by the **open** and **createWindow** methods, and is updated when **CurrentWindow** is set or **clearText**, **displayTextAt**, or **destroyWindow** is called. It is also updated when **displayText** is called if **CursorUpdate** is true.

- Errors** A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	An invalid cursor column value was specified.

- See Also** **CursorRow** Property, **displayText** Method.

CursorRow Property

- Syntax** **CursorRow: *int32* { read-write, access after open }**
- Remarks** Holds the row in the current window to which the next displayed character will be output.
- Legal values range from zero through one less than **Rows**.
- This property is initialized to zero by the **open** and **createWindow** methods, and is updated when **CurrentWindow** is set or **clearText**, **displayTextAt**, or **destroyWindow** is called. It is also updated when **displayText** is called if **CursorUpdate** is true.

- Errors** A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	An invalid cursor row value was specified.

- See Also** **CursorColumn** Property, **displayText** Method.

CursorType Property**Updated in Release 1.8**

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

Remarks Holds the cursor type for the current window. The following are the possible values:

Value	Meaning
DISP_CT_NONE	Cursor is not displayed.
DISP_CT_BLOCK	Cursor is displayed as a block.
DISP_CT_HALFBLOCK	Cursor is displayed as a halfblock.
DISP_CT_UNDERLINE	Cursor is displayed as an underline.
DISP_CT_REVERSE	Cursor is displayed in reverse video.
DISP_CT_BLINK	A blinking cursor is supported. This value is to be logically ORed with one of the other values defined for this property.
DISP_CT_OTHER	Cursor is displayed but form is unknown.

This property cannot be written if **CapCursorType** has either DISP_CCT_NONE or DISP_CCT_FIXED set. Otherwise it can be set to one of the cursor types specified by **CapCursorType**, and if supported, DISP_CT_BLINK can be logically ORed with that cursor type to display a blinking cursor.

This property is maintained for each window. Setting this property affects only the current window since only the current window has a displayable cursor.

This property is initialized to DISP_CT_NONE (or the appropriate cursor type if **CapCursorType** has DISP_CCT_FIXED set) by the **open** and **createWindow** methods, and is updated when **CurrentWindow** is set or **destroyWindow** is called.

Errors A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	CapCursorType is either DISP_CCT_NONE or DISP_CCT_FIXED is set, or an invalid cursor type value was specified.

See Also **CapCursorType** Property.

CursorUpdate Property

Syntax **CursorUpdate: *boolean* { read-write, access after open }**

Remarks When true, **CursorRow** and **CursorColumn** will be updated to point to the character beyond the last character output when characters are displayed using the **displayText** or **displayTextAt** method. When false, the cursor properties will not be updated when characters are displayed.

This property is maintained for each window. It initialized to true by the **open** and **createWindow** methods, and is updated when **CurrentWindow** is set or **destroyWindow** is called.

Errors A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

See Also **CursorRow** Property, **CursorColumn** Property.

CustomGlyphList Property**Added in Release 1.6**

Syntax	CustomGlyphList: <i>string</i> { read-only, access after open }
Remarks	<p>Contains character codes that are available for definition as glyphs. Character codes are represented as two-digit (ASCII) or four-digit (Unicode) hexadecimal values. These values are comma separated and each value may actually represent a range of values specified by using the '-' character.</p> <p>For example, if the string is "2D,4D", then the device supports glyph definitions for the characters "-" and "M" respectively. If the string is "002D-004D", then the device supports glyph definitions for the Unicode characters between 002D and 004D inclusive. Also, if the string is "2D-2F,3D-3F", then the device supports glyph definitions for the ranges of hex characters 2D through 2F and 3D through 3F.</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 16.
See Also	CapCustomGlyph Property, GlyphHeight Property, GlyphWidth Property, defineGlyph Method.

DeviceBrightness Property

Syntax	DeviceBrightness: <i>int32</i> { read-write, access after open-claim-enable }				
Remarks	<p>Holds the device brightness value, expressed as a percentage between 0 and 100.</p> <p>Any device can support 0% (blank) and 100% (full intensity). Blanking can, at a minimum, be supported by sending spaces to the device. If CapBrightness is true, then the device also supports one or more levels of dimming.</p> <p>If a device does not support the specified brightness value, then the Service will choose an appropriate substitute.</p> <p>This property is initialized to 100 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 16.</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 value was used: Not in the range 0 - 100.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	An invalid value was used: Not in the range 0 - 100.
Value	Meaning				
E_ILLEGAL	An invalid value was used: Not in the range 0 - 100.				
See Also	CapBrightness Property.				

DeviceColumns Property

Updated in Release 1.7

Syntax	DeviceColumns: <i>int32</i> { read-only, access after open }
Remarks	Holds the number of columns on this device. This property is initialized by the open method. It is updated when the ScreenMode property is changed.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	DeviceRows Property, ScreenMode Property.

DeviceDescriptors Property

Syntax	DeviceDescriptors: <i>int32</i> { read-only, access after open }
Remarks	Holds the number of descriptors on this device. If CapDescriptors is true, then this property is non-zero. 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 16.
See Also	setDescriptor Method, clearDescriptors Method.

DeviceRows Property

Updated in Release 1.7

Syntax	DeviceRows: <i>int32</i> { read-only, access after open }
Remarks	Holds the number of rows on this device. This property is initialized by the open method. It is updated when the ScreenMode property is changed.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	DeviceColumns Property, ScreenMode Property.

DeviceWindows Property

Syntax	DeviceWindows: <i>int32</i> { read-only, access after open }
Remarks	Holds the maximum window number supported by this device. A value of zero indicates that only the device window is supported and that no windows may be created. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	CurrentWindow Property.

GlyphHeight Property**Added in Release 1.6**

- Syntax** **GlyphHeight:** *int32* { read-only, access after open }
- Remarks** Indicates the glyph height based on the number of pixels for a character cell. 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 16.
- See Also** **CapCustomGlyph** Property, **CustomGlyphList** Property, **defineGlyph** Method.

GlyphWidth Property**Added in Release 1.6**

- Syntax** **GlyphWidth:** *int32* { read-only, access after open }
- Remarks** Indicates the glyph width based on the number of pixels for a character cell. 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 16.
- See Also** **CapCustomGlyph** Property, **CustomGlyphList** Property, **defineGlyph** Method.

InterCharacterWait Property

- Syntax** **InterCharacterWait:** *int32* { read-write, access after open }
- Remarks** Holds the wait time between displaying each character with the **displayText** and **displayTextAt** methods. This provides a “teletype” appearance when displaying text.
- This property is only used if the window is not in *Marquee Mode* — that is, **MarqueeType** must be `DISP_MT_NONE`.
- When non-zero and the window is not in *Marquee Mode*, the window is in *Teletype Mode*: **displayText** and **displayTextAt** requests are enqueued and processed in the order they are received. This property specifies the time to wait between outputting each character into the viewport. The wait time is the specified number of milliseconds. (Note that the system timer resolution may reduce the precision of the wait time.) If **CursorUpdate** is true, **CursorRow** and **CursorColumn** are updated to their final values before **displayText** or **displayTextAt** returns, even though all of its data may not yet be displayed.
- When this property is zero and the window is not in *Marquee Mode*, *Immediate Mode* is in effect, so that characters are processed as quickly as possible. If some display requests are enqueued at the time this property is set to zero, the requests are completed as quickly as possible.
- If **CapICharWait** is false, then intercharacter waiting is not supported, and the value of this property is not used.
- This property is initialized to zero by the **open** and **createWindow** methods, and is updated when **CurrentWindow** is set or **destroyWindow** is called.
- Errors** A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
- Some possible values of the exception’s *ErrorCode* property are:
- | Value | Meaning |
|------------------------|---------------------------------|
| <code>E_ILLEGAL</code> | An illegal value was specified. |
- See Also** **displayText** Method.

MapCharacterSet Property**Added in Release 1.7**

- Syntax** **MapCharacterSet:** *boolean* { read-write, access after open }
- Remarks** If **MapCharacterSet** is true and when outputting data, the Service maps the characters transferred by the application to the character set selected in the **CharacterSet** property for displaying data.
- If **MapCharacterSet** is false, then no mapping is supported. In such a case the application has to ensure the mapping of the character set used in the application to the character set selected in the **CharacterSet** property.
- If **CapMapCharacterSet** is false, then this property is always 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 16.
- See Also** **CharacterSet** Property, **CapMapCharacterSet** Property.

MarqueeFormat Property

- Syntax** **MarqueeFormat:** *int32* { read-write, access after open }
- Remarks** Holds the marquee format for the current window.
- | Value | Meaning |
|---------------|--|
| DISP_MF_WALK | Begin the marquee by walking data from the opposite side. For example, if the marquee type is “left,” then the viewport is filled by bringing characters into the right side and scrolling them to the left. |
| DISP_MF_PLACE | Begin the marquee by placing data. For example, if the marquee type is “left,” then the viewport is filled by placing characters starting at the left side, and beginning scrolling only after the viewport is full. |

This property is initialized to DISP_MF_WALK by the **open** and **createWindow** methods, and is updated when **CurrentWindow** is set or **destroyWindow** is called.

This property is read when a transition is made to **Marquee On Mode**. It is not used when not in **Marquee Mode**.

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

1. Map the window to the viewport as follows:

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

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

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

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

1. Map the window to the viewport as follows:

<u>Marquee TypeWindow</u>	=	<u>Viewport</u>
LeftFirst Column	=	First Column
UpFirst Row	=	First Row
RightLast Column	=	Last Column
DownLast Row	=	Last Row

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

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

Errors

A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	An invalid value was used, or attempted to change window 0.

See Also

MarqueeType Property, **MarqueeUnitWait** Property, **MarqueeRepeatWait** Property.

Example 1

Marquee Walk format.

- Assume a 2x20 display.
- An application has a line display instance named myLD.
- The application has performed:


```
myLD.createWindow(0, 3, 2, 3, 2, 5); // 2x3 viewport of 2x5 window
myLD.displayText("0123456789", DISP_DT_NORMAL);
```

The window contains:

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

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

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

If the application performs the sequence:

```
myLD.setMarqueeType(DISP_MT_INIT);
myLD.setMarqueeFormat(DISP_MF_WALK);
myLD.displayTextAt(0, 4, "AB", DISP_DT_NORMAL);
```

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

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

If the application performs:

```
myLD.setMarqueeType(DISP_MT_LEFT);
```

the window is not changed, and the viewport becomes:

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

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

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

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

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

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

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

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

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

The marquee has scrolled to the end of the window.

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

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

Example 2 Marquee Place format.

- Assume a 2x20 display.
- An application has a line display instance named myLD.
- The application has performed:
`myLD.createWindow(0, 3, 2, 3, 2, 5); // 2x3 viewport of 2x5 window`
`myLD.displayText("0123456789", DISP_DT_NORMAL);`

The window contains:

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

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

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

If the application performs the sequence:

```
myLD.setMarqueeType(DISP_MT_INIT);
myLD.setMarqueeFormat(DISP_MF_PLACE);
myLD.displayTextAt(0, 4, "AB", DISP_DT_NORMAL);
```

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

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

If the application performs:

```
myLD.setMarqueeType(DISP_MT_LEFT);
```

the window is not changed, and the viewport becomes:

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

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

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

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

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

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

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

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

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

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

The marquee has scrolled to the end of the window.

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

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

MarqueeRepeatWait Property

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

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

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

This property is initialized to zero by the **open** and **createWindow** methods, and is updated when **CurrentWindow** is set or **destroyWindow** is called.

This property is not used if not in *Marquee Mode*.

Errors A `UposException` may be thrown when this property is accessed. For further information, see "Errors" on page 16.

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

Value	Meaning
E_ILLEGAL	An illegal value was specified.

See Also **MarqueeType** Property, **MarqueeFormat** Property, **MarqueeUnitWait** Property.

MarqueeType Property

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

Remarks Holds the marquee type for the current window. When not DISP_MT_NONE, the window is in *Marquee Mode*. This property has one of the following values:

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

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

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

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

This property is always DISP_MT_NONE for window 0 – the device window.

This property is initialized to DISP_MT_NONE by the **open** and **createWindow** methods, and is updated when **CurrentWindow** is set or **destroyWindow** is called.

Errors A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	An invalid value was used, or attempted to change window 0.

See Also **MarqueeFormat** Property, **MarqueeUnitWait** Property, **MarqueeRepeatWait** Property.

MarqueeUnitWait Property

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

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

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

This property is not used if **MarqueeType** is DISP_MT_NONE.

This property is initialized to zero by the **open** and **createWindow** methods, and is updated when **CurrentWindow** is set or **destroyWindow** is called.

Errors A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	An illegal value was specified.

See Also **MarqueeType** Property, **MarqueeFormat** Property, **MarqueeRepeatWait** Property.

MaximumX Property

Added in Release 1.7

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

Remarks A value of zero indicates that bitmaps are not supported. Otherwise, contains the maximum number of horizontal pixels supported by the device. It must be less than 65,536. Dividing **MaximumX** by **DeviceColumns** gives the number of pixels required for each character.

This property is initialized by the **open** method. It may be updated when the **ScreenMode** property is changed.

Errors A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

See Also **DeviceColumns** Property, **ScreenMode** Property, **MaximumY** Property.

MaximumY Property***Added in Release 1.7***

- Syntax** **MaximumY:** *int32* { read-only, access after open }
- Remarks** A value of zero indicates that bitmaps are not supported. Otherwise, contains the maximum number of vertical pixels supported by the device. It must be less than 65,536. Dividing **MaximumY** by **DeviceRows** gives the number of pixels required for each character
- This property is initialized by the **open** method. It may be updated when the **ScreenMode** property is changed.
- Errors** A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
- See Also** **DeviceRows** Property, **MaximumX** Property, **ScreenMode** Property.

Rows Property

- Syntax** **Rows:** *int32* { read-only, access after open }
- Remarks** Holds the number of rows for this window.
- For window 0, this property is the same as **DeviceRows**.
For other windows, it may be less or greater than **DeviceRows**.
- This property is initialized to **DeviceRows** by the **open** method, and is updated when **CurrentWindow** is set or **createWindow** or **destroyWindow** are called.
- Errors** A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
- See Also** **Columns** Property.

ScreenMode Property***Added in Release 1.7***

- Syntax** **ScreenMode:** *int32* { read-write, access after open-claim }
- Remarks** Contains the screen mode value of the device. If **CapScreenMode** is false, then only a value of zero is allowed. If **CapScreenMode** is true, then the values can be set to index the values contained in **ScreenModeList**. For example:
- 0 = Default value
 1 = First setting in **ScreenModeList**
 2 = Second setting in **ScreenModeList**, etc.
- Note:** This property can only be updated when the device is opened and claimed, but **not** enabled.
- Changing the **ScreenMode** property also changes the **DeviceColumns** and **DeviceRows** properties to the new screen size. Also, for some devices, the **MaximumX** and **MaximumY** properties may be changed due to the columns and/or rows requiring a different number of physical pixels. For example, if the display physically contains 48x256 pixels and supports 2x20, 4x32, and 5x32, then the Service layout may be:

Mode	Pixels per Row	Pixels per Column	MaximumY	MaximumX	Unused Vertical Pixels	Unused Horizontal Pixels
2x20	24	12	48	240	0	16
4x32	12	8	48	256	0	0
5x32	8	8	40	256	8	0

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 16.

See Also **CapScreenMode** Property, **DeviceColumns** Property, **DeviceRows** Property, **MaximumX** Property, **MaximumY** Property, **ScreenModeList** Property.

ScreenModeList Property

Added in Release 1.7

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

Remarks Contains the comma-delimited list of row-column pairs that are supported by the device.

If **CapScreenMode** is false, only one pair will be listed. For example, if the device only supports 2 rows and 20 columns, then this property should be set to “2x20”.

If the device can operate in 2 by 20, 4 by 32, or 5 by 32 modes, then this property should be set to “2x20,4x32,5x32”.

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 16.

See Also **CapScreenMode** Property, **ScreenMode** Property.

Methods (UML operations)

clearDescriptors Method

Syntax	clearDescriptors (): void { raises-exception, use after open-claim-enable }				
Remarks	Turns off all descriptors. This function is illegal if CapDescriptors is false.				
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16. Some possible values of the exception’s <i>ErrorCode</i> property are:				
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>E_ILLEGAL</td> <td>The device does not support descriptors.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	The device does not support descriptors.
Value	Meaning				
E_ILLEGAL	The device does not support descriptors.				
See Also	setDescriptor Method, DeviceDescriptors Property, CapDescriptors Property.				

clearText Method

Updated in Release 1.7

Syntax	clearText (): void { raises-exception, use after open-claim-enable }				
Remarks	Clears the current window to blanks, sets CursorRow and CursorColumn to zero, and resynchronizes the beginning of the window with the start of the viewport. All clears all bitmaps displayed in the window. If in <i>Immediate Mode</i> or <i>Teletype Mode</i> , the viewport is also cleared immediately. If in <i>Marquee Init Mode</i> , the viewport is not changed. If in <i>Marquee On Mode</i> , this method is illegal.				
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16. Some possible values of the exception’s <i>ErrorCode</i> property are:				
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>E_ILLEGAL</td> <td>In <i>Marquee On Mode</i>.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	In <i>Marquee On Mode</i> .
Value	Meaning				
E_ILLEGAL	In <i>Marquee On Mode</i> .				
See Also	displayText Method.				

createWindow Method**Updated in Release 1.6**

Syntax **createWindow (viewportRow: int32, viewportColumn: int32, viewportHeight: int32, viewportWidth: int32, windowHeight: int32, windowWidth: int32): void { raises-exception, use after open-claim-enable }**

Parameter	Description
<i>viewportRow</i>	The viewport's start device row.
<i>viewportColumn</i>	The viewport's start device column.
<i>viewportHeight</i>	The number of device rows in the viewport.
<i>viewportWidth</i>	The number of device columns in the viewport.
<i>windowHeight</i>	The number of rows in the window.
<i>windowWidth</i>	The number of columns in the window.

Remarks Creates a viewport over the portion of the display given by the first four parameters. The window size is given by the last two parameters. Valid window row values range from zero to one less than *windowHeight* and column values range from zero to one less than *windowWidth*.

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

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

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

Property	Value
Rows	Set to <i>windowHeight</i> .
Columns	Set to <i>windowWidth</i> .
CursorRow	Set to 0.
CursorColumn	Set to 0.
CursorType	Set to DISP_CT_NONE (or the appropriate cursor type if CapCursorType has DISP_CCT_FIXED set).
CursorUpdate	Set to true.
MarqueeType	Set to DISP_MT_NONE.
MarqueeFormat	Set to DISP_MF_WALK.
MarqueeUnitWait	Set to 0.
MarqueeRepeatWait	Set to 0.
InterCharacterWait	Set to 0.

Errors A UpoException may be thrown when this method is invoked. For further information, see "Errors" on page 16.

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

Value	Meaning
E_ILLEGAL	One or more parameters are out of their valid ranges, or all available windows are already in use.

See Also **CapCursorType** Property, **CurrentWindow** Property, **destroyWindow** Method.

defineGlyph Method

Updated in Release 1.7

Syntax **defineGlyph (glyphCode: *int32*, glyph: *binary*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>glyphCode</i>	The character code to be defined.
<i>glyph</i>	Data bytes that define the glyph. ¹

Remarks Defines a glyph character.

The glyph is defined as bits representing each pixel packed into bytes using whole bytes to represent each row.

The minimum number of bytes are sent for each row, based on **GlyphWidth** and using 8 bits per byte. Bytes are sent left-to-right across each row; if more than one byte is required per row, the leftmost byte is sent first. The lowest-order bit within a byte represents the rightmost pixel. Bits that do not represent pixels are the highest order bits and their value is ignored. Rows are sent from the top down.

A 10 pixel wide glyph would have the two leftmost pixels represented in bits 1 and 0 of the first byte, respectively. The remaining 8 pixels would be represented in the second byte.

Enough rows must be sent to define the entire character. Whether changing the definition of a glyph causes currently displayed characters to change, or the change appears only when next drawn, is hardware-defined.

Example: A 5 column 7 row character cell –

Bit Position 76543210	Byte	Hex Value
. * . . .	0	08
. . * . .	1	04
* . . * .	2	12
. * . . *	3	09
. . * . .	4	04
. . . * .	5	02
. . . . *	6	01

¹. In the OPOS environment, the format of this data depends upon the value of the **BinaryConversion** property. See **BinaryConversion** property on page A-28.

Example: A 12 column by 16 row character cell –

Bit Position	Bytes	Hex Values
111111		
5432109876543210		

.....	0,1	00 00
.....*	2,3	00 40
....***	4,5	00 E0
...**.**	6,7	01 B0
..**.**	8,9	03 18
..**.**	10,11	03 18
..*****	12,13	03 F8
..*****	14,15	03 F8
..**.**	16,17	03 18
..**.**	18,19	03 18
..**.**	20,21	03 18
.....	22,23	00 00
.....	24,25	00 00
.....	26,27	00 00
.....	28,29	00 00
.....	30,31	00 00

This function is illegal if **CapCustomGlyph** is false.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	CapCustomGlyph is false, or <i>glyphCode</i> is an unsupported character code for glyph definition.

See Also **CapCustomGlyph** Property, **CustomGlyphList** Property, **GlyphHeight** Property, **GlyphWidth** Property.

destroyWindow Method

Syntax	destroyWindow (): void { raises-exception, use after open-claim-enable }				
Remarks	Destroys the current window. The characters displayed in its viewport are not changed. CurrentWindow is set to window 0. The device window and the associated window properties are updated.				
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16. Some possible values of the exception’s <i>ErrorCode</i> property are:				
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>E_ILLEGAL</td> <td>The current window is 0. This window may not be destroyed.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	The current window is 0. This window may not be destroyed.
Value	Meaning				
E_ILLEGAL	The current window is 0. This window may not be destroyed.				
See Also	createWindow Method, CurrentWindow Property.				

displayBitmap Method**Added in Release 1.7**

Syntax	displayBitmap (fileName:string,width:int32,alignmentX:int32,alignmentY:int32): void { raises-exception, use after open-claim-enable }																								
	<table> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><i>fileName</i></td> <td>File name or URL of bitmap file. Various file formats may be supported, such as bmp, gif, or jpeg files.²</td> </tr> <tr> <td><i>width</i></td> <td>Width of the bitmap to be displayed. See values below.</td> </tr> <tr> <td><i>alignmentX</i></td> <td>Horizontal placement of the bitmap. See values below.</td> </tr> <tr> <td><i>alignmentY</i></td> <td>Vertical placement of the bitmap. See values below.</td> </tr> </tbody> </table> <p>The <i>width</i> parameter has one of the following values:</p> <table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>DISP_BM_ASIS</td> <td>Display the bitmap with one bitmap pixel per dot.</td> </tr> <tr> <td>Other values</td> <td>Bitmap width expressed in number of pixels.</td> </tr> </tbody> </table> <p>The <i>alignmentX</i> parameter has one of the following values:</p> <table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>DISP_BM_LEFT</td> <td>Align the bitmap's left edge with the leftmost pixel of the current character position, as specified by CursorColumn.</td> </tr> <tr> <td>DISP_BM_CENTER</td> <td>Align the bitmap in the horizontal center of the current character position, as specified by CursorColumn.</td> </tr> <tr> <td>DISP_BM_RIGHT</td> <td>Align the bitmap's right edge with the rightmost pixel of the current character position, as specified by CursorColumn.</td> </tr> </tbody> </table>	Parameter	Description	<i>fileName</i>	File name or URL of bitmap file. Various file formats may be supported, such as bmp, gif, or jpeg files. ²	<i>width</i>	Width of the bitmap to be displayed. See values below.	<i>alignmentX</i>	Horizontal placement of the bitmap. See values below.	<i>alignmentY</i>	Vertical placement of the bitmap. See values below.	Value	Meaning	DISP_BM_ASIS	Display the bitmap with one bitmap pixel per dot.	Other values	Bitmap width expressed in number of pixels.	Value	Meaning	DISP_BM_LEFT	Align the bitmap's left edge with the leftmost pixel of the current character position, as specified by CursorColumn .	DISP_BM_CENTER	Align the bitmap in the horizontal center of the current character position, as specified by CursorColumn .	DISP_BM_RIGHT	Align the bitmap's right edge with the rightmost pixel of the current character position, as specified by CursorColumn .
Parameter	Description																								
<i>fileName</i>	File name or URL of bitmap file. Various file formats may be supported, such as bmp, gif, or jpeg files. ²																								
<i>width</i>	Width of the bitmap to be displayed. See values below.																								
<i>alignmentX</i>	Horizontal placement of the bitmap. See values below.																								
<i>alignmentY</i>	Vertical placement of the bitmap. See values below.																								
Value	Meaning																								
DISP_BM_ASIS	Display the bitmap with one bitmap pixel per dot.																								
Other values	Bitmap width expressed in number of pixels.																								
Value	Meaning																								
DISP_BM_LEFT	Align the bitmap's left edge with the leftmost pixel of the current character position, as specified by CursorColumn .																								
DISP_BM_CENTER	Align the bitmap in the horizontal center of the current character position, as specified by CursorColumn .																								
DISP_BM_RIGHT	Align the bitmap's right edge with the rightmost pixel of the current character position, as specified by CursorColumn .																								

². In the OPOS environment, the Service Object must support two-color (black and white) uncompressed Windows bitmaps. Black pixels are displayed with the foreground color, while white pixels are displayed with the background color. Additional formats may be supported.

Other values Distance from the window's leftmost pixel column to the left edge of the bitmap, expressed in number of pixels.

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

Value	Meaning
DISP_BM_TOP	Align the bitmap's top edge with the topmost pixel of the current character position, as specified by CursorRow .
DISP_BM_CENTER	Align the bitmap in the vertical center of the current character position, as specified by CursorRow .
DISP_BM_BOTTOM	Align the bitmap's bottom edge with the bottommost pixel of the current character position, as specified by CursorRow .

Other values Distance from the window's topmost pixel row to the start of the bitmap, expressed in number of pixels.

Remarks Called to display a bitmap on the LineDisplay. The bitmap is displayed within the current window's viewport.

If DISP_BM_... constants are specified for *alignmentX* and *alignmentY*, then it is displayed in relation to the character position specified by **CursorRow** and **CursorColumn**. If, in addition, **CursorUpdate** is true, then **CursorRow** and **CursorColumn** are updated to point to the first character position following the bitmap.

If the bitmap does not exactly occupy a multiple of rows and columns, then the unoccupied pixels of those character positions which are partially occupied are displayed with the background color. In other words, the Service will effectively fill all character positions partially or completely occupied by the bitmap with the background color before drawing the bitmap.

Bitmap display has the following restrictions:

- Bitmap display is only legal in **Immediate Mode**.
- The window size must match the window's viewport size.
- The bitmap must be displayable within the window, after consideration of the function parameters. For example, if *alignmentX* specifies a pixel near the bottom of the window, and the bitmap height (after bitmap transformation, if required) exceeds the distance from *alignmentX* to the window bottom, then the bitmap is not displayed.

The *width* parameter controls transformation of the bitmap. If *width* is DISP_BM_ASIS, then no transformation is performed. The bitmap is displayed with one bitmap pixel per line display pixel. The advantages of this option are that it:

- Provides the highest performance bitmap display.
- Works well for bitmaps tuned for a specific LineDisplay's aspect ratio between horizontal 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. The advantages of this option are that it:

- Sizes a bitmap to fit a variety of LineDisplays.
- Maintains the bitmap's aspect ratio.

The disadvantages of this option are:

- Lower performance than untransformed data.
- Some lines and images that are “smooth” in the original bitmap may show some “ratcheting”.

Errors A `UposException` may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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"> • The <code>LineDisplay</code> does not support bitmap display (CapBitmap is false). • The <i>width</i> parameter is invalid or too big. • The <i>alignmentX</i> / <i>alignmentY</i> parameter is invalid or too big. • The window is not in Immediate Mode. • The window size does not match its viewport size. • The bitmap is too large to display at the requested location.
E_NOEXIST	The <i>fileName</i> was not found.
E_EXTENDED	<i>ErrorCodeExtended</i> = <code>EDISP_TOOBIG</code> : The bitmap is either too wide to display without transformation, or it is too big to transform. <i>ErrorCodeExtended</i> = <code>EDISP_BADFORMAT</code> : The specified file is either not a bitmap file or it is an unsupported format.

See Also **CapBitmap** Property, **CursorColumn** Property, **CursorRow** Property, **CursorUpdate** Property.

displayText Method

Updated in Release 1.7

Syntax `displayText (data: string, attribute: int32): void { raises-exception, use after open-claim-enable }`

Parameter	Description
<i>data</i>	The string of characters to display. ³
<i>attribute</i>	The display attribute for the text. Must be either <code>DISP_DT_NORMAL</code> , <code>DISP_DT_BLINK</code> , <code>DISP_DT_REVERSE</code> , or <code>DISP_DT_BLINK_REVERSE</code> .

Remarks The characters in *data* are processed beginning at the location specified by **CursorRow** and **CursorColumn**, and continue in succeeding character positions. Any previous data in a character position is overwritten, including character and bitmap data.

Character processing continues to the next row when the end of a window row is reached. If the end of the window is reached with additional characters to be processed, then the window is scrolled upward by one row and the bottom row is

³. In the **OPOS** environment, the format of this data depends upon the value of the **BinaryConversion** property. See **BinaryConversion** property on page A-28.

set to blanks. If **CursorUpdate** is true, then **CursorRow** and **CursorColumn** are updated to point to the character position following the last character of *data*.

Note

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

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

The operation of **displayText** (and **displayTextAt**) varies for each mode:

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

If **CapBlink** is DISP_CB_NOBLINK, then *attribute* value DISP_DT_BLINK is ignored, and *attribute* DISP_DT_BLINK_REVERSE is treated as DISP_DT_REVERSE. If **CapBlink** is DISP_CB_BLINKALL, then the entire display will blink when one or more characters have been set to blink. If **CapBlink** is DISP_CB_BLINKEACH, then only those characters displayed with the blink *attribute* will blink.

If **CapReverse** is DISP_CR_NONE, then *attribute* value DISP_DT_REVERSE is ignored, and *attribute* DISP_DT_BLINK_REVERSE is treated as DISP_DT_BLINK. If **CapReverse** is DISP_CR_REVERSEALL, then the entire display will be displayed in reverse video when one or more characters have been set to reverse. If **CapReverse** is DISP_CR_REVERSEEACH, then only those characters displayed with the reverse *attribute* will be displayed in reverse video.

The *attribute* parameter value establishes the initial blink and reverse video *attributes*. Beginning with Release 1.7, escape sequences within *data* may be used to set or reset these *attributes*.

Special character values within *data* are:

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

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	<i>attribute</i> is illegal, or the display is in Marquee On Mode .

See Also **CapBlink** Property, **CapReverse** Property, **CursorColumn** Property, **CursorRow** Property, **CursorUpdate** Property, **InterCharacterWait** Property, **clearText** Method, **displayTextAt** Method.

displayTextAt Method

Updated in Release 1.7

Syntax `displayTextAt (row: int32, column: int32, data: string, attribute: int32): void { raises-exception, use after open-claim-enable }`

Parameter	Description
<i>row</i>	The start row for the text.
<i>column</i>	The start column for the text.
<i>data</i>	The string of characters to display. ⁴
<i>attribute</i>	The display attribute for the text. Must be either DISP_DT_NORMAL, DISP_DT_BLINK, DISP_DT_REVERSE, or DISP_DT_BLINK_REVERSE.

Remarks The characters in *data* are processed beginning at the window location specified by the *row* and *column* parameters, and continuing in succeeding columns. The operational characteristics of the **displayTextAt** method are the same as the **displayText** method.

This method has the same effect as setting the **CursorRow** to *row*, setting **CursorColumn** to *column*, and calling the **displayText** method.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	<i>row</i> or <i>column</i> are out of range, <i>attribute</i> is illegal, or in Marquee On Mode .

See Also **CapBlink** Property, **CapReverse** Property, **CursorColumn** Property, **CursorRow** Property, **InterCharacterWait** Property, **displayText** Method, **clearText** Method.

⁴. In the OPOS environment, the format of this data depends upon the value of the **BinaryConversion** property. See **BinaryConversion** property on page A-28.

readCharacterAtCursor Method**Added in Release 1.6**

Syntax	readCharacterAtCursor (inout cursorData: <i>int32</i>): void { raises-exception, use after open-claim-enable }				
	<table> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><i>cursorData</i></td> <td>The character read from the display.</td> </tr> </tbody> </table>	Parameter	Description	<i>cursorData</i>	The character read from the display.
Parameter	Description				
<i>cursorData</i>	The character read from the display.				
Remarks	Reads the currently displayed character at the cursor position. This function is illegal if CapReadBack is DISP_CRB_NONE.				
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16. Some possible values of the exception’s <i>ErrorCode</i> property are:				
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>E_ILLEGAL</td> <td>CapReadBack is DISP_CRB_NONE.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	CapReadBack is DISP_CRB_NONE.
Value	Meaning				
E_ILLEGAL	CapReadBack is DISP_CRB_NONE.				
See Also	CapReadBack Property.				

refreshWindow Method

Syntax	refreshWindow (window: <i>int32</i>): void { raises-exception, use after open-claim-enable }				
	The <i>window</i> parameter specifies which window must be refreshed.				
Remarks	Changes the current window to <i>window</i> , then redisplay its viewport. Neither the mapping of the window to its viewport nor the window’s cursor position is changed. This function may be used to restore a window after another window has overwritten some of its viewport.				
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16. Some possible values of the exception’s <i>ErrorCode</i> property are:				
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>E_ILLEGAL</td> <td><i>window</i> is larger than DeviceWindows or has not been created, or in <i>Marquee On Mode</i>.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	<i>window</i> is larger than DeviceWindows or has not been created, or in <i>Marquee On Mode</i> .
Value	Meaning				
E_ILLEGAL	<i>window</i> is larger than DeviceWindows or has not been created, or in <i>Marquee On Mode</i> .				
See Also	DeviceWindows Property.				

scrollText Method**Updated in Release 1.7**

Syntax **scrollText (direction: *int32*, units: *int32*):**
 void { raises-exception, use after open-claim-enable }

The *direction* parameter indicates the scrolling direction, and is one of the following values:

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

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

Remarks Scrolls the current window.

This function is only legal in **Immediate Mode**.

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

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

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	<i>direction</i> is illegal, or in Teletype Mode or Marquee Mode .

See Also **displayText Method.**

Example 1 - Assume a 2x20 display.
 - An application has a line display instance named myLD.
 - The application has performed:
 myLD.createWindow(0, 3, 2, 4, 2, 4); // 2x4 viewport of 2x4 window
 myLD.displayText(“abcdABCD”, DISP_DT_NORMAL);

The window contains:

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

and the viewport on the display is:

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

If the application next performs:

```
myLD.scrollText (DISP_ST_LEFT, 2);
```

the window data becomes:

	0	1	2	3
0	c	d		
1	C	D		

and the viewport becomes:

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

Example 2

- Assume a 2x20 display.
- An application has a line display instance named myLD.
- The application has performed:

```
myLD.createWindow(0, 3, 2, 4, 2, 8); // 2x4 viewport of 2x8 window
```

```
myLD.displayText("abcdefghABCDEFGH", DISP_DT_NORMAL);
```

The window contains:

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

and the viewport on the display is:

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

If the application next performs:

```
myLD.scrollText (DISP_ST_LEFT, 2);
```

the window data is unchanged, and the viewport becomes:

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

If the application next performs:
`myLD.scrollText (DISP_ST_UP, 1);`
 the window data becomes:

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

and the viewport becomes:

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

setBitmap Method

Added in Release 1.7

Syntax `setBitmap (bitmapNumber: int32, fileName: string, width: int32, alignmentX: int32, alignmentY: int32);`
void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>bitmapNumber</i>	The number to be assigned to this bitmap. Valid bitmap numbers are 1 through 100.
<i>fileName</i>	File name or URL of bitmap file. Various file formats may be supported, such as bmp, gif, or jpeg files. ⁵
<i>width</i>	If set to the empty string (“”), then the bitmap is unset. Width of the bitmap to be displayed. See values below.
<i>alignmentX</i>	Horizontal placement of the bitmap. See values below.
<i>alignmentY</i>	Vertical placement of the bitmap. See values below.

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

Value	Meaning
DISP_BM_ASIS	Display the bitmap with one bitmap pixel per dot.
Other values	Bitmap width expressed in number of pixels.

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

Value	Meaning
DISP_BM_LEFT	Align the bitmap’s left edge with the leftmost pixel of the current character position.
DISP_BM_CENTER	Align the bitmap in the horizontal center of the current character position.
DISP_BM_RIGHT	Align the bitmap’s right edge with the rightmost pixel of the current character position.

⁵. In the OPOS environment, the Service Object must support two-color (black and white) uncompressed Windows bitmaps. Black pixels are displayed with the foreground color, while white pixels are displayed with the background color. Additional formats may be supported.

Other values Distance from the window's leftmost pixel column to the left edge of the bitmap, expressed in number of pixels.

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

Value	Meaning
DISP_BM_TOP	Align the bitmap's top edge with the topmost pixel of the current character position.
DISP_BM_CENTER	Align the bitmap in the vertical center of the current character position.
DISP_BM_BOTTOM	Align the bitmap's bottom edge with the bottommost pixel of the current character position.
Other values	Distance from the window's topmost pixel row to the start of the bitmap, expressed in number of pixels.

Remarks Called to save information about a bitmap for later display.

The bitmap may then be displayed by calling the **displayText** or **displayTextAt** method with the display bitmap escape sequence in the display data. The display bitmap escape sequence will typically be included in a string for displaying advertisements, store logos, or icons. See the Remarks section of **displayBitmap** for restrictions on displaying the saved bitmap. If one or more restrictions are not fulfilled, then the bitmap is not displayed, and the method continues on with the next character of display data.

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 most frequently used bitmaps should be assigned a small *bitmapNumber* (close to 1), while occasionally used bitmaps should be assigned the larger *bitmapNumbers*. The Service will use this information to determine how best to store the bitmaps. It may download them to the device when possible, or cache them in Service memory, or simply remember the *fileName* and associated properties for use when it is displayed.

An application must ensure that the LineDisplay window metrics, such as viewport width and height, are set before calling this method. A Service may perform transformations on the bitmap in preparation for later displaying based on the current values of these metrics.

Errors A `UposException` may be thrown when this method is invoked. For further information, see "Errors" on page 16.

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"> The <i>bitmapNumber</i> parameter is invalid. The LineDisplay does not support bitmap display (CapBitmap is false). The <i>width</i> parameter is invalid or too big. The <i>alignmentX</i> or <i>alignmentY</i> parameter is invalid or too big.

E_NOEXIST	The <i>fileName</i> was not found.
E_EXTENDED	<i>ErrorCodeExtended</i> = EDISP_TOOBIG: The bitmap is either too wide to display without transformation, or it is too big to transform. <i>ErrorCodeExtended</i> = EDISP_BADFORMAT: The specified file is either not a bitmap file or it is an unsupported format.

See Also **CapBitmap** Property, **displayBitmap** Method, **displayText** Method, **displayTextAt** Method.

setDescriptor Method

Syntax **setDescriptor (descriptor: *int32*, attribute: *int32*):**
void { raises-exception, use after open-claim-enable }

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

The *attribute* parameter indicates the attribute for the descriptor. It has one of the following values:

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

Remarks Sets the state of one of the descriptors, which are small indicators with a fixed label.
This function is illegal if **CapDescriptors** is false.
The device and its Service determine the mapping of *descriptor* to its descriptors.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	The device does not support descriptors, or one of the parameters contained an illegal value.

See Also **clearDescriptors** Method, **DeviceDescriptors** Property, **CapDescriptors** 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 Line Display 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 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 Line Display devices which may not have any knowledge of the Service's need for this event.

See Also "Events" on page 15, **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 a Line Display.

Attributes This event contains the following attribute:

Attribute	Type	Description
<i>Status</i>	<i>int32</i>	Reports a change in the power state of a display.

Note that Release 1.3 added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See "StatusUpdateEvent" description on page 63.

Remarks Enqueued when the Line Display detects a power state change.

See Also "Events" on page 15.

MICR - Magnetic Ink Character Recognition Reader

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

Summary

Properties (UML attributes)

<i>Common</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
AutoDisable:	<i>boolean</i>	{ read-write }	1.2	open
CapPowerReporting:	<i>int32</i>	{ read-only }	1.3	open
CapStatisticsReporting:	<i>boolean</i>	{ read-only }	1.8	open
CapUpdateStatistics:	<i>boolean</i>	{ read-only }	1.8	open
CheckHealthText:	<i>string</i>	{ read-only }	1.0	open
Claimed:	<i>boolean</i>	{ read-only }	1.0	open
DataCount:	<i>int32</i>	{ read-only }	1.2	open
DataEventEnabled:	<i>boolean</i>	{ read-write }	1.0	open
DeviceEnabled:	<i>boolean</i>	{ read-write }	1.0	open & claim
FreezeEvents:	<i>boolean</i>	{ read-write }	1.0	open
OutputID:	<i>int32</i>	{ read-only }	1.0	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 }	1.0	--
DeviceControlDescription:	<i>string</i>	{ read-only }	1.0	--
DeviceControlVersion:	<i>int32</i>	{ read-only }	1.0	--
DeviceServiceDescription:	<i>string</i>	{ read-only }	1.0	open
DeviceServiceVersion:	<i>int32</i>	{ read-only }	1.0	open
PhysicalDeviceDescription:	<i>string</i>	{ read-only }	1.0	open
PhysicalDeviceName:	<i>string</i>	{ read-only }	1.0	open

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 }	1.0	open
Amount:	<i>string</i>	{ read-only }	1.0	open
BankNumber:	<i>string</i>	{ read-only }	1.0	open
CapValidationDevice:	<i>boolean</i>	{ read-only }	1.0	open
CheckType:	<i>int32</i>	{ read-only }	1.0	open
CountryCode:	<i>int32</i>	{ read-only }	1.0	open
EPC:	<i>string</i>	{ read-only }	1.0	open
RawData:	<i>string</i>	{ read-only }	1.0	open
SerialNumber:	<i>string</i>	{ read-only }	1.0	open
TransitNumber:	<i>string</i>	{ read-only }	1.0	open

Methods (UML operations)**Common**

<i>Name</i>	<i>Version</i>
open (logicalDeviceName: <i>string</i>): void { raises-exception }	1.0
close (): void { raises-exception, use after open }	1.0
claim (timeout: <i>int32</i>): void { raises-exception, use after open }	1.0
release (): void { raises-exception, use after open, claim }	1.0
checkHealth (level: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.0
clearInput (): void { raises-exception, use after open, claim }	1.0
clearOutput (): void { }	<i>Not supported</i>
directIO (command: <i>int32</i>, inout data: <i>int32</i>, inout obj: <i>object</i>): void { raises-exception, use after open }	1.0
resetStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
retrieveStatistics (inout statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
updateStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8

Specific*Name*

beginInsertion (timeout: int32): void { raises-exception, use after open, claim, enable }	1.0
beginRemoval (timeout: int32): void { raises-exception, use after open, claim, enable }	1.0
endInsertion (): void { raises-exception, use after open, claim, enable }	1.0
endRemoval (): void { raises-exception, use after open, claim, enable }	1.0

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>
upos::events::DataEvent			1.0
Status:	<i>int32</i>	{ read-only }	
upos::events::DirectIOEvent			1.0
EventNumber:	<i>int32</i>	{ read-only }	
Data:	<i>int32</i>	{ read-write }	
Obj:	<i>object</i>	{ read-write }	
upos::events::ErrorEvent			1.0
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		<i>Not Supported</i>	
upos::events::StatusUpdateEvent			1.3
Status:	<i>int32</i>	{ read-only }	

General Information

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

Capabilities

The MICR Control has the following minimal set of capabilities:

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

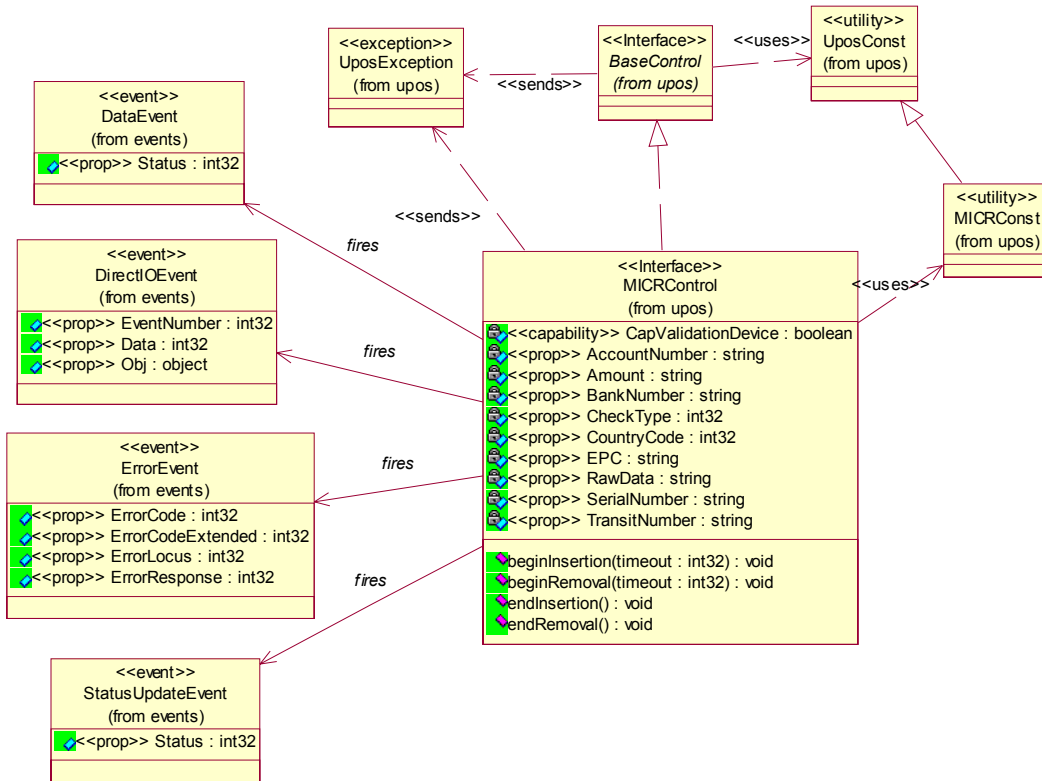
The MICR device may be physically attached to or incorporated into a check validation print device. If this is the case, once a check is inserted via MICR Control methods, the check can still be used by the Printer Control prior to check removal.

Some MICR devices support exception tables, which cause non-standard parsing of the serial number for specific check routing numbers. Exception tables are not directly supported by this specification release. However, a Service may choose to support them, and could assign entries under its device name to define the exception entries.

This release of the specification does not define any parsing of partial MICR check data if an error occurs while reading a check. This is done intentionally since any Service that implements such functionality cannot guarantee that fields parsed from partial data are correct. For example, it is possible to get MICR data that contains no ‘?’ characters, but fails its checksum. This would indicate that one or more characters in the data are incorrect, but there is no way to determine which characters they are. If an application wishes to try and parse the partial data itself, the **RawData** property is filled in with whatever was read even in error cases.

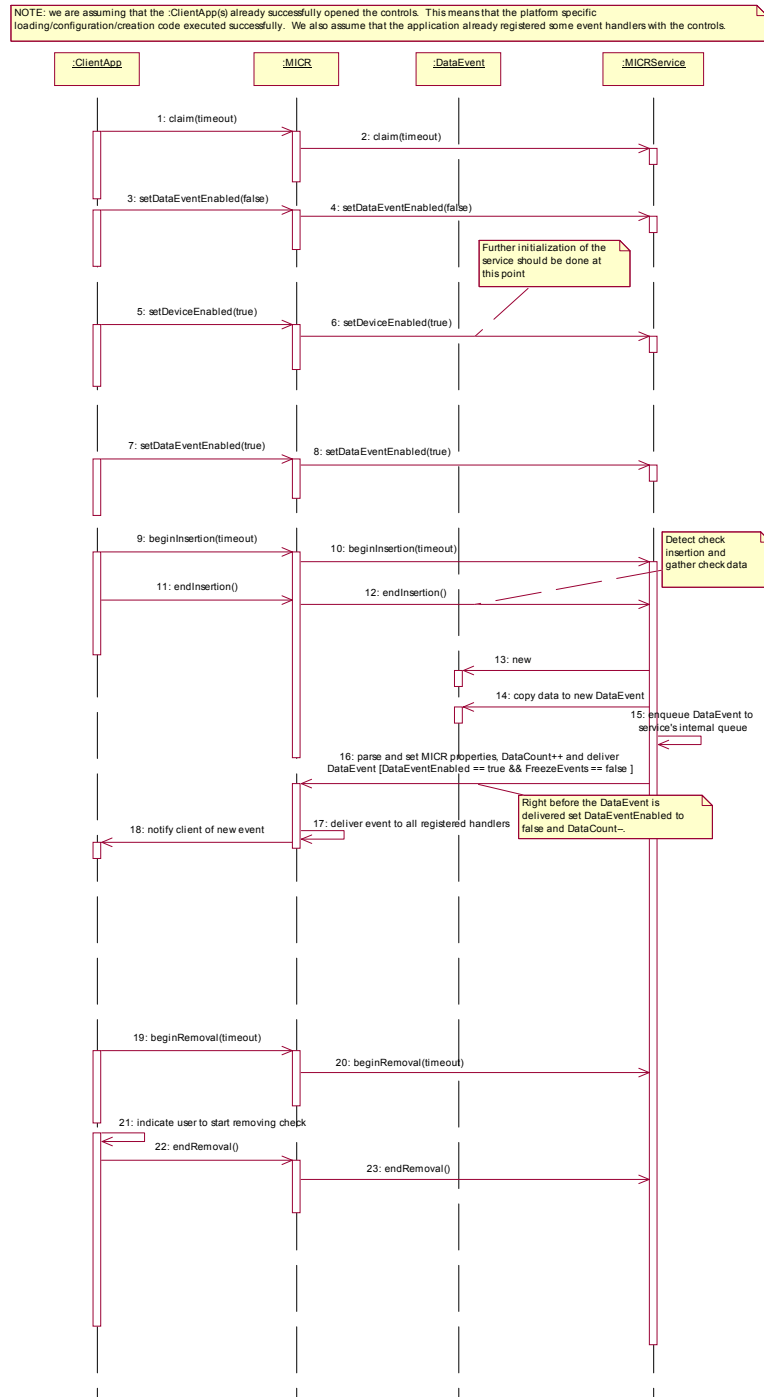
MICR Class Diagram

The following diagram shows the relationships between the MICR classes.



MICR Sequence Diagram *Updated in Release 1.8*

The following sequence diagram shows the typical usage of the MICR device. This also demonstrate the usage of the “Device Input Model” and how that works with **DataEventEnabled**; also shows the steps in the check removal process.



Model

The MICR Device follows the general “Device Input Model” for input devices. One point of difference is that the MICR Device requires the invocation of methods to insert and remove the check for processing. Therefore, this Device requires more than simply setting the **DataEventEnabled** property to true in order to receive data. The basic model is as follows:

- The MICR Control is opened, claimed, and enabled.
- When an application wishes to perform a MICR read, the application calls **beginInsertion**, specifying a timeout value. This results in the device being made ready to have a check inserted. If the check is not inserted before the timeout limit expires, a `UposException` is raised.

In the event of a timeout, the MICR device will remain in a state allowing a check to be inserted while the application provides any additional prompting required and then reissues the **beginInsertion** method.
- Once a check is inserted, the method returns and the application calls **endInsertion**, which results in the MICR device being taken out of check insertion mode and the check, if present, actually being read.
 - If the check is successfully read, a **DataEvent** is enqueued.
 - If the **AutoDisable** property is true, then the Device automatically disables itself when a **DataEvent** is enqueued.
 - A queued **DataEvent** can be delivered to the application when **DataEventEnabled** is true and other event delivery requirements are met. Just before delivering this event, data is copied into properties, and further data events are disabled by setting **DataEventEnabled** to false. This causes subsequent input data to be enqueued while the application processes the current input and associated properties. When the application has finished processing the current input and is ready for more data, it reenables events by setting **DataEventEnabled** to true.
 - An **ErrorEvent** (or events) is enqueued if an error occurs while reading the check, and is delivered to the application when **DataEventEnabled** is true and other event delivery requirements are met.
 - The **DataCount** property may be read to obtain the number of enqueued **DataEvents**.
 - All enqueued input may be deleted by calling **clearInput**. See the **clearInput** method description for more details.
- After processing a **DataEvent**, the application should query the **CapValidationDevice** property to determine if validation printing can be performed on the check prior to check removal. If this property is true, the application may call the Printer Service’s **beginInsertion** and **endInsertion** methods. This positions the check for validation printing. The POS Printer’s validation printing methods can then be used to perform validation printing. When validation printing is complete, the application should call the Printer Service’s removal methods to remove the check.

- Once the check is no longer needed in the device, the application must call the **beginRemoval** method of the MICR, or the Check Scanner (if the device can also scan checks), or the POS Printer (if **CapValidationDevice** is true), specifying a timeout value. This method will raise a `UposException` if the check is not removed within the timeout period. In this case, the application may perform any additional prompting prior to calling the method again. Once the check is removed, the application should call the same device's **endRemoval** method to take the device out of removal mode.

Many models of MICR devices do not require any check handling processing from the application. Such MICR devices may always be capable of processing a check and require no commands to actually read and eject the check. For these types of MICR devices, the **beginInsertion**, **endInsertion**, **beginRemoval**, and **endRemoval** methods simply return, and input data will be enqueued until the **DataEventEnabled** property is set to true. However, applications should still use these methods to ensure application portability across different MICR devices.





Device Sharing

The MICR is an exclusive-use device. Its device sharing rules are:

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

MICR Character Substitution

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

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

Properties (UML attributes)

AccountNumber Property

Syntax	AccountNumber: <i>string</i> { read-only, access after open }
Remarks	<p>Holds the account number parsed from the most recently read MICR data.</p> <p>This account number will not include a check serial number if a check serial number is able to be separately parsed, even if the check serial number is embedded in the account number portion of the ‘On Us’ field. If the account number cannot be identified, the string will be empty (“”).</p> <p>Its value is set prior to a DataEvent being delivered to the application.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	RawData Property, DataEvent .

Amount Property

Syntax	Amount: <i>string</i> { read-only, access after open }
Remarks	<p>Holds the amount field parsed from the most recently read MICR data.</p> <p>The amount field on a check consists of ten digits bordered by Amount symbols. All non space digits will be represented in the test string including leading 0’s. If the amount is not present, the string will be empty (“”).</p> <p>Its value is set prior to a DataEvent being delivered to the application.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	RawData Property, DataEvent .

BankNumber Property

Syntax	BankNumber: <i>string</i> { read-only, access after open }
Remarks	<p>Holds the bank number portion of the transit field parsed from the most recently read MICR data.</p> <p>The bank number is contained in digits 5 through 8 of the transit field. If the bank number or transit field is not present or successfully identified, the string will be empty (“”).</p> <p>Its value is set prior to a DataEvent being delivered to the application.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	RawData Property, TransitNumber Property, DataEvent .

CapValidationDevice Property

- Syntax** **CapValidationDevice: *boolean* { read-only, access after open }**
- Remarks** If true, the device also performs validation printing via the POS Printer's slip station, and a check does not have to be removed from the MICR device prior to performing validation printing.
- For devices that are both a MICR device as well as a POS Printer, the device will automatically position the check for validation printing after successfully performing a MICR read. Either the MICR's or the POS Printer's **beginRemoval** and **endRemoval** methods may be called to remove the check once processing is complete.
- 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 16.

CheckType Property

- Syntax** **CheckType: *int32* { read-only, access after open }**
- Remarks** Holds the type of check parsed from the most recently read MICR data. It has one of the following values:
- | Value | Meaning |
|------------------|--|
| MICR_CT_PERSONAL | The check is a personal check. |
| MICR_CT_BUSINESS | The check is a business or commercial check. |
| MICR_CT_UNKNOWN | Unknown type of check. |
- Its value is set prior to a **DataEvent** being delivered to the application.
- Errors** A UposException may be thrown when this property is accessed. For further information, see "Errors" on page 16.
- See Also** **RawData** Property, **DataEvent**.

CountryCode Property

Syntax	CountryCode: <i>int32</i> { read-only, access after open }										
Remarks	Holds the country of origin of the check parsed from the most recently read MICR data. It has one of the following values:										
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>MICR_CC_USA</td> <td>The check is from America.</td> </tr> <tr> <td>MICR_CC_CANADA</td> <td>The check is from Canada.</td> </tr> <tr> <td>MICR_CC_MEXICO</td> <td>The check is from Mexico.</td> </tr> <tr> <td>MICR_CC_UNKNOWN</td> <td>Check origination is unknown.</td> </tr> </tbody> </table>	Value	Meaning	MICR_CC_USA	The check is from America.	MICR_CC_CANADA	The check is from Canada.	MICR_CC_MEXICO	The check is from Mexico.	MICR_CC_UNKNOWN	Check origination is unknown.
Value	Meaning										
MICR_CC_USA	The check is from America.										
MICR_CC_CANADA	The check is from Canada.										
MICR_CC_MEXICO	The check is from Mexico.										
MICR_CC_UNKNOWN	Check origination is unknown.										
	Its value is set prior to a DataEvent being delivered to the application.										
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.										
See Also	RawData Property, DataEvent .										

EPC Property

Syntax	EPC: <i>string</i> { read-only, access after open }
Remarks	Holds the Extended Processing Code (“EPC”) field parsed from the most recently read MICR data. It will contain a single character 0 though 9 if the field is present. If not, the string will be empty (“”).
	Its value is set prior to a DataEvent being delivered to the application.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	RawData Property, DataEvent .

RawData Property

Syntax	RawData: <i>string</i> { read-only, access after open }
Remarks	Holds the MICR data from the most recent MICR read. It contains any of the 15 MICR characters with appropriate substitution to represent non-ASCII characters (see “MICR Character Substitution”, page 459). No parsing or special processing is done to the data returned in this property. A sample value may look like the following: `"2t123456789t123 4 567890o 123 a0000001957a" Note that spaces are used to represent spaces in the MICR data. Its value is set prior to a DataEvent being delivered to the application.
Errors	A <code>UposException</code> may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	AccountNumber Property, Amount Property, BankNumber Property, CheckType Property, CountryCode Property, EPC Property, SerialNumber Property, TransitNumber Property, DataEvent .

SerialNumber Property

Syntax	SerialNumber: <i>string</i> { read-only, access after open }
Remarks	Holds the serial number of the check parsed from the most recently read MICR data. If the serial number cannot be successfully parsed, the string will be empty (“”). Its value is set prior to a DataEvent being delivered to the application.
Errors	A <code>UposException</code> may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	RawData Property, DataEvent .

TransitNumber Property

Syntax	TransitNumber: <i>string</i> { read-only, access after open }
Remarks	Holds the transit field of the check parsed from the most recently read MICR data. It consists of all the characters read between the ‘Transit’ symbols on the check. It is a nine character string. Its value is set prior to a DataEvent being delivered to the application.
Errors	A <code>UposException</code> may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	RawData Property, DataEvent .

Methods (UML operations)

beginInsertion Method

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

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

If zero, the method tries to begin insertion mode, then returns immediately if successful. Otherwise a UposException is raised. If UPOS_FOREVER (-1), the method initiates the begin insertion mode, then waits as long as needed until either the check is inserted or an error occurs.

Remarks Initiates check insertion processing.

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

If the MICR device cannot be placed into insertion mode, a UposException is raised. Otherwise, check insertion is monitored until either:

- The check is successfully inserted.
- The check is not inserted before *timeout* milliseconds have elapsed, or an error is reported by the MICR device. In this case, a UposException is raised. The MICR device remains in check insertion mode. This allows an application to perform some user interaction and reissue the **beginInsertion** method without altering the MICR check handling mechanism.

Errors A UposException may be thrown when this method is invoked. For further information, see "Errors" on page 16.

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

Value	Meaning
E_BUSY	If the MICR is a combination device, the peer device may be busy.
E_ILLEGAL	An invalid <i>timeout</i> parameter was specified.
E_TIMEOUT	The specified time has elapsed without the check 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 }

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

If zero, the method tries to begin removal mode, then returns immediately if successful. Otherwise a `UposException` is raised. If `UPOS_FOREVER (-1)`, the method initiates the begin removal mode, then waits as long as needed until either the check is removed or an error occurs.

Remarks Initiates check removal processing.

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

If the MICR device cannot be placed into removal or ejection mode, a `UposException` is raised. Otherwise, check removal is monitored until either:

- The check is successfully removed.
- The check is not removed before *timeout* milliseconds have elapsed, or an error is reported by the MICR device. In this case, a `UposException` is raised. The MICR device remains in check removal mode. This allows an application to perform some user interaction and reissue the **beginRemoval** method without altering the MICR check handling mechanism.

Errors A `UposException` may be thrown when this method is invoked. For further information, see "Errors" on page 16.

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

Value	Meaning
E_BUSY	If the MICR is a combination device, the peer device may be busy.
E_ILLEGAL	An invalid <i>timeout</i> parameter was specified.
E_TIMEOUT	The specified time has elapsed without the check being properly removed.

See Also **beginInsertion** Method, **endInsertion** Method, **endRemoval** Method.

endInsertion Method

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

Remarks Ends check insertion processing.

When called, the MICR is taken out of check insertion mode. If a check is not detected in the device, a `UposException` is raised with an extended error code of `EMICR_NOCHECK`. After a successful **endInsertion**, if a check is detected, the check will be read by the MICR device and either a **DataEvent** or **ErrorEvent** will be delivered. Data will be available as soon as the **DataEventEnabled** property is set to true. This allows an application to prompt the user prior to calling this method to ensure that the form is correctly positioned.

This method is paired with the **beginInsertion** method for controlling check insertion. Although some MICR devices do not require this sort of processing, the application should still use these methods to ensure application portability across different MICR devices.

Errors A `UposException` may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	The device is not in check insertion mode.
E_EXTENDED	<i>ErrorCodeExtended</i> = <code>EMICR_NOCHECK</code> : The device was taken out of insertion mode without a check being inserted.

See Also **beginInsertion** Method, **beginRemoval** Method, **endRemoval** Method.

endRemoval Method

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

Remarks Ends check removal processing.

When called, the MICR is taken out of check removal or ejection mode. If a check is detected in the device, a `UposException` is raised with an extended error code of `EMICR_CHECK`.

This method is paired with the **beginRemoval** method for controlling check removal. Although some MICR devices do not require this sort of processing, the application should still use these methods to ensure application portability across different MICR devices.

Errors A `UposException` may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
<code>E_ILLEGAL</code>	The device is not in check removal mode.
<code>E_EXTENDED</code>	<i>ErrorCodeExtended</i> = <code>EMICR_CHECK</code> : The device was taken out of removal mode while a check is still present.

See Also **beginInsertion** Method, **endInsertion** Method, **beginRemoval** Method.

Events (UML interfaces)

DataEvent

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

Description Notifies the application when MICR data is read from a check and is available to be read.

Attributes This event contains the following attribute:

Attribute	Type	Description
<i>Status</i>	<i>int32</i>	Set to zero.

Before delivering this event, the **RawData** property is updated and the data is parsed (if possible) into the MICR data fields.

See Also “Device Input Model” on page 18, “Events” on page 15, **RawData** Property, **AccountNumber** Property, **Amount** Property, **BankNumber** Property, **CheckType** Property, **CountryCode** Property, **EPC** Property, **SerialNumber** Property, **TransitNumber** 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 MICR 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 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 as part of the JavaPOS standard. Use of this event may restrict the application program from being used with other vendor’s MICR devices which may not have any knowledge of the Service’s need for this event.

See Also “Events” on page 15, **directIO** Method.

ErrorEvent**Updated in Release 1.7**

```

<< 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 when reading MICR data.

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> on page 16.
<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* contains one of the following values:

Value	Meaning
EMICR_BADDATA	An unreadable character was detected during input processing. The RawData property will contain partial data if available, otherwise it will be an empty string.
EMICR_NODATA	The entire input data stream was unreadable. No data is available.
EMICR_BADSIZE	The length of the check was beyond the expected readable range. The RawData property will contain partial data if available, otherwise it will be an empty string.
EMICR_JAM	The check insertion process has caused a paper jam. No data is available.
EMICR_CHECKDIGIT	The check digit verification has failed even though there was no error during input processing. The RawData property will contain partial data if available, otherwise it will be an empty string.
EMICR_COVEROPEN	The check insertion process failed due to the POSPrinter cover being open. No data is available.

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

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 DataEventEnabled is again set to true, then another ErrorEvent is delivered with locus EL_INPUT. Default when locus is EL_INPUT_DATA.

- Remarks** This event is not delivered until **DataEventEnabled** is true and other event delivery requirements are met, so that proper application sequencing occurs.
- See Also** “Device Input Model” on page 18, “Device Information Reporting Model” on page 26.

StatusUpdateEvent

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

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

Attributes This event contains the following attribute:

Attribute	Type	Description
<i>Status</i>	<i>int32</i>	Reports a change in the power state of a MICR device.

Note that Release 1.3 added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See “StatusUpdateEvent” description on page 63.

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

See Also “Events” on page 15.

Motion Sensor

This Chapter defines the Motion Sensor device category.

Summary

Properties (UML attributes)

<i>Common</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
AutoDisable:	<i>boolean</i>	{ read-write }	1.7	Not Supported
CapPowerReporting:	<i>int32</i>	{ read-only }	1.7	open
CapStatisticsReporting:	<i>boolean</i>	{ read-only }	1.8	open
CapUpdateStatistics:	<i>boolean</i>	{ read-only }	1.8	open
CheckHealthText:	<i>string</i>	{ read-only }	1.7	open
Claimed:	<i>boolean</i>	{ read-only }	1.7	open
DataCount:	<i>int32</i>	{ read-only }	1.7	Not Supported
DataEventEnabled:	<i>boolean</i>	{ read-write }	1.7	Not Supported
DeviceEnabled:	<i>boolean</i>	{ read-write }	1.7	open
FreezeEvents:	<i>boolean</i>	{ read-write }	1.7	open
OutputID:	<i>int32</i>	{ read-only }	1.7	Not Supported
PowerNotify:	<i>int32</i>	{ read-write }	1.7	open
PowerState:	<i>int32</i>	{ read-only }	1.7	open
State:	<i>int32</i>	{ read-only }	1.7	--
DeviceControlDescription:	<i>string</i>	{ read-only }	1.7	--
DeviceControlVersion:	<i>int32</i>	{ read-only }	1.7	--
DeviceServiceDescription:	<i>string</i>	{ read-only }	1.7	open
DeviceServiceVersion:	<i>int32</i>	{ read-only }	1.7	open
PhysicalDeviceDescription:	<i>string</i>	{ read-only }	1.7	open
PhysicalDeviceName:	<i>string</i>	{ read-only }	1.7	open
<i>Specific</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
Timeout:	<i>int32</i>	{ read-write }	1.7	open & enable
Motion:	<i>boolean</i>	{ read-only }	1.7	open & enable

Methods (UML operations)**Common**

<i>Name</i>	<i>Version</i>
open (logicalDeviceName: <i>string</i>): void { raises-exception }	1.7
close (): void { raises-exception, use after open }	1.7
claim (timeout: <i>int32</i>): void { raises-exception, use after open }	1.7
release (): void { raises-exception, use after open, claim }	1.7
checkHealth (level: <i>int32</i>): void { raises-exception, use after open, enable }	1.7
clearInput (): void { }	<i>Not supported</i>
clearOutput (): void { }	<i>Not supported</i>
directIO (command: <i>int32</i>, inout data: <i>int32</i>, inout obj: <i>object</i>): void { raises-exception, use after open }	1.7
resetStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
retrieveStatistics (inout statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
updateStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8

Specific

<i>Name</i>	<i>Version</i>
waitForMotion(timeout: <i>int32</i>): void { raises-exception, use after open, enable }	1.7

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>
upos::events::DataEvent		<i>Not Supported</i>	
upos::events::DirectIOEvent			1.7
EventNumber:	<i>int32</i>	{ read-only }	
Data:	<i>int32</i>	{ read-write }	
Obj:	<i>object</i>	{ read-write }	
upos::events::ErrorEvent		<i>Not Supported</i>	
upos::events::OutputCompleteEvent		<i>Not Supported</i>	
upos::events::StatusUpdateEvent			1.7
Status:	<i>int32</i>	{ read-only }	

General Information

The Motion Sensor programmatic name is “MotionSensor”.

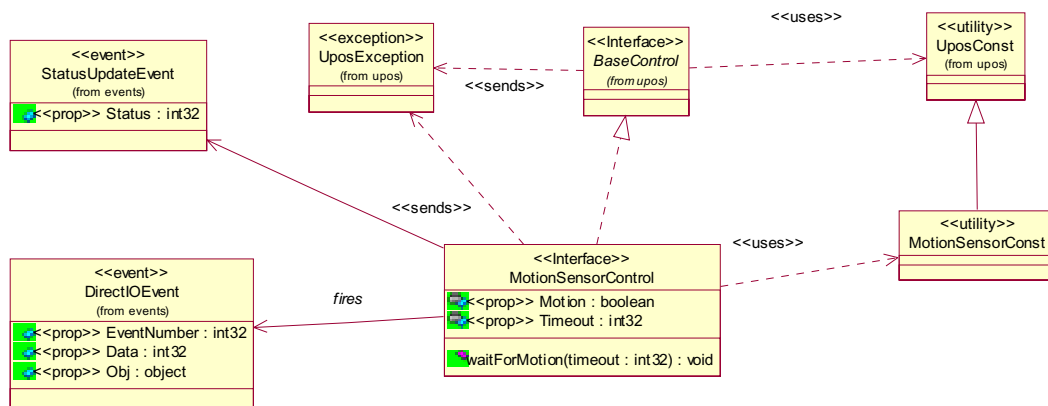
Capabilities

The Motion Sensor has the following minimal set of capabilities:

- Supports detection of person present at POS device
- Supports reporting of motion detection changes, either by hardware or software detection.

Motion Sensor Class Diagram

The following diagram shows the relationships between the Motion Sensor classes.



Model

The Motion Sensor defines two Motion Sensor indications as constants. It is assumed that the Motion Sensor supports present and absent indications. The constants for these Motion Sensor positions and their values are as follows:

- MOTION_M_PRESENT 1
- MOTION_M_ABSENT 2

StatusUpdateEvents are fired using the above values. The **Timeout** value is used to set the number of milliseconds between the last time someone was present and a MOTION_M_ABSENT **StatusUpdateEvent** being fired.

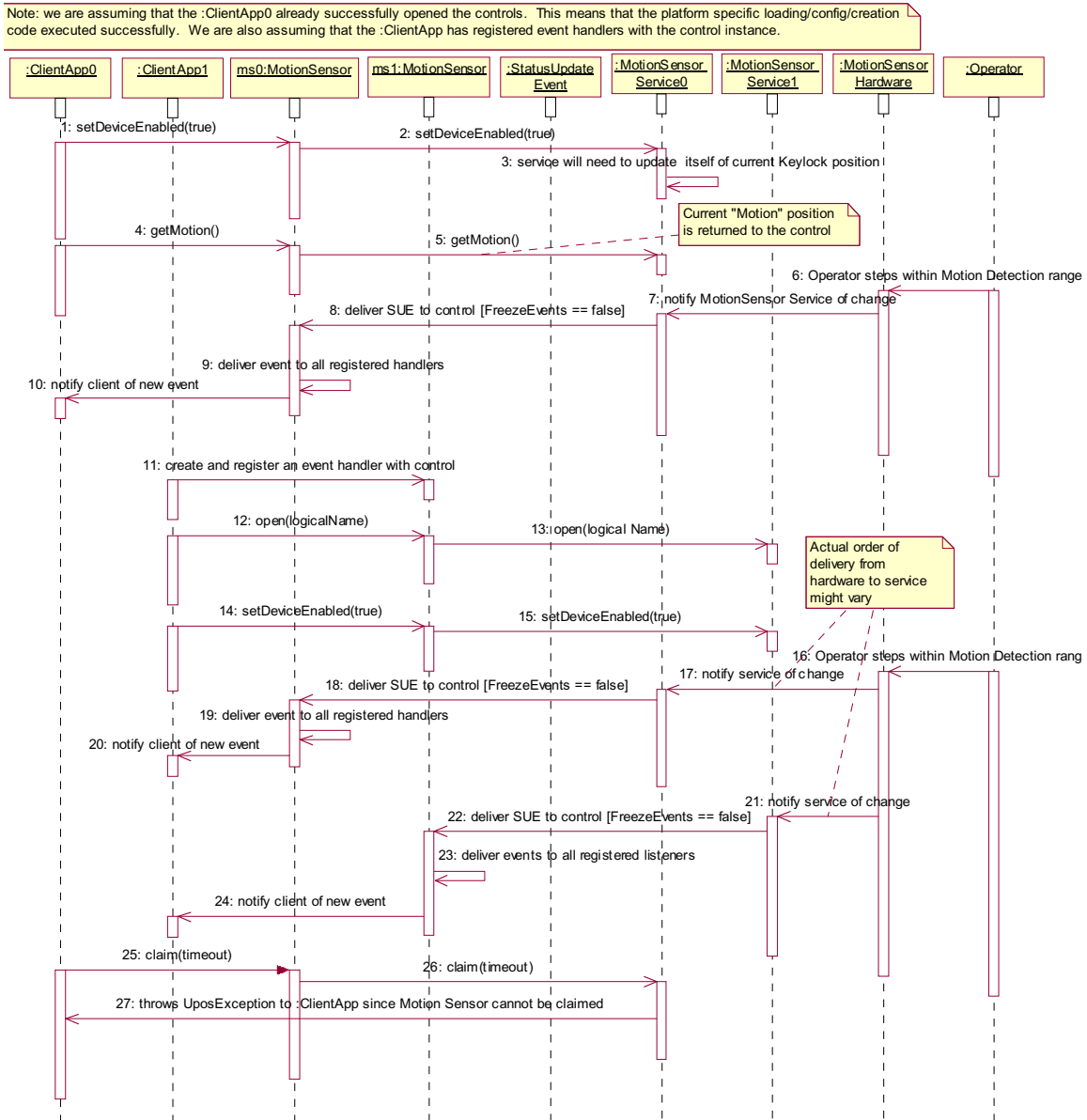
Device Sharing

The Motion Sensor is a sharable device. Its device sharing rules are:

- After opening and enabling the device, the application may access all properties and methods and will receive status update events.
- If more than one application has opened and enabled the device, each of these applications may access its properties and methods. Status update events are fired to all of these applications.
- The Motion Sensor may not be claimed for exclusive access. Therefore, if an application calls **claim** or **release**, these methods will always raise a **UposException**.
- See the “Summary” table for precise usage prerequisites.

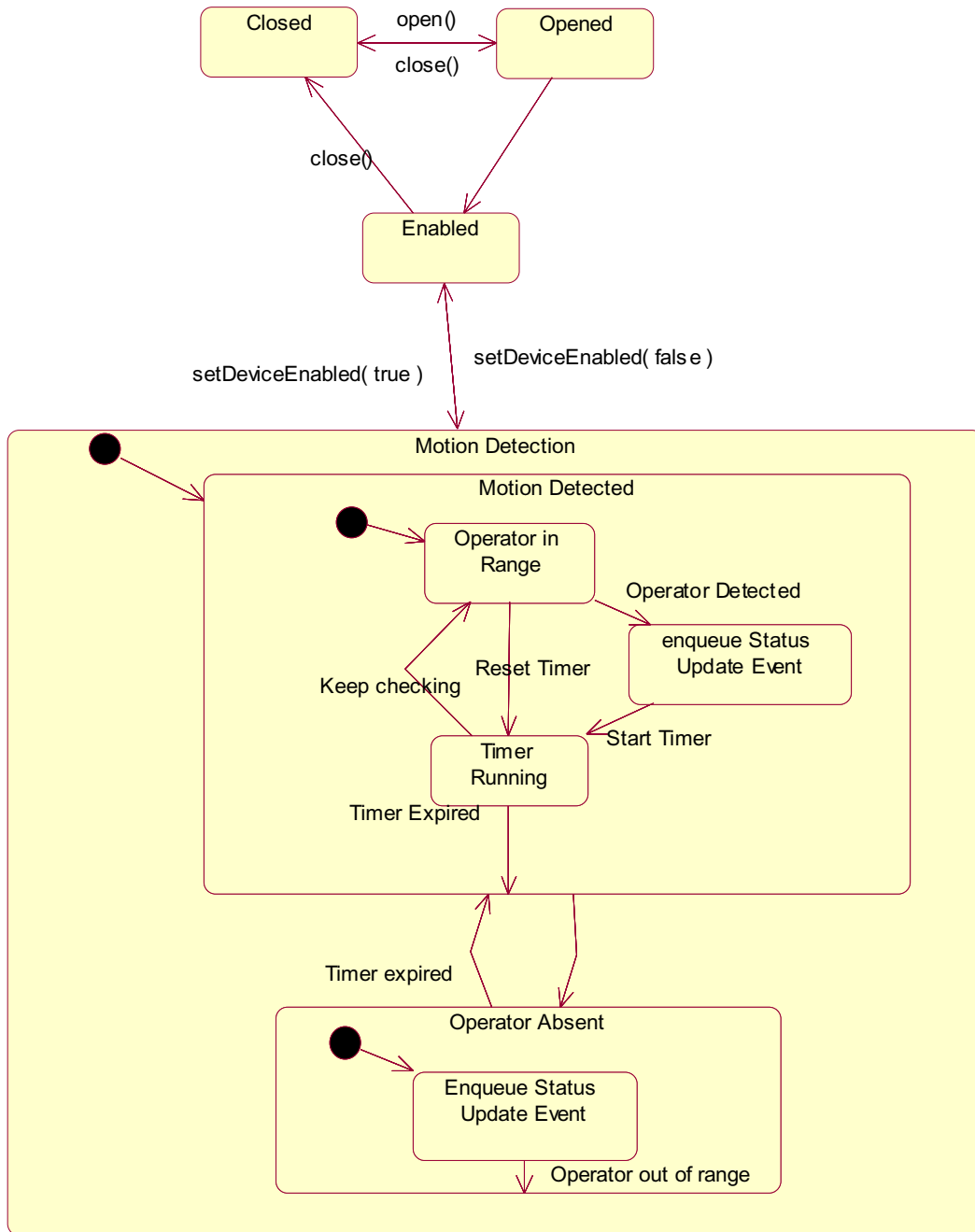
Motion Sensor Sequence Diagram

The following sequence diagram shows the typical usage of the Motion Sensor device.



Motion Sensor State Diagram

The following state diagram depicts the Motion Sensor Control device model.



Properties (UML attributes)

Motion Property

Syntax	Motion: <i>boolean</i> { read-only, access after open-enable }
Remarks	Holds a boolean value that indicates whether motion has been detected. 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 16.

Timeout Property

Syntax	Timeout: <i>int32</i> { read-write, access after open-enable }
Remarks	<p>Holds a value that indicates the number of milliseconds from the last time motion was detected until the StatusUpdateEvent of MOTION_M_ABSENT is fired.</p> <p>This property needs to be application specific for a shared device. If several applications are sharing the device, each application may set an independent timeout value, and each application will receive StatusUpdateEvents according to its supplied timeout.</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 16.
See Also	StatusUpdateEvent.

Methods (UML operations)

waitForMotion Method

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

Parameter	Description
<i>timeout</i>	Maximum number of milliseconds for the Motion Sensor to wait for a person to be present before returning control back to the application. If zero, the method returns immediately. If UPOS_FOREVER (-1), the method waits as long as needed until motion is detected or an error occurs.

Remarks Waits for a presence detection from the Motion Sensor.
If the Motion Sensor detects someone is present, then the method returns immediately.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_TIMEOUT	The <i>timeout</i> period expired before motion was detected.

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

See Also "Events" on page 15, **directIO** Method.

StatusUpdateEvent

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

Description Notifies the application when the Motion Sensor detects a change.

Attributes This event contains the following attribute:

Attribute	Type	Description
-----------	------	-------------

<i>Status</i>	<i>int32</i>	The status of the Motion Sensor.
---------------	--------------	----------------------------------

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

Value	Description
-------	-------------

MOTION_M_PRESENT	Motion Sensor has detected someone is present. Value is one (1).
------------------	--

MOTION_M_ABSENT	Motion Sensor has detected no one has been present for the number of milliseconds specified in Timeout . Value is two (2).
-----------------	---

Note that Release 1.3 added Power State Reporting with additional *Power reporting* **StatusUpdateEvent** values. See “StatusUpdateEvent” description on page 63.

Remarks This event is enqueued when a Motion Sensor detection undergoes a change or if Power State Reporting is enabled and a change in the power state is detected.

See Also **Timeout** Property, “Events” on page 15.

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</i>	<i>May Use After</i>
AutoDisable:	<i>boolean</i>	{ read-write }	1.2	open
CapPowerReporting:	<i>int32</i>	{ read-only }	1.3	open
CapStatisticsReporting:	<i>boolean</i>	{ read-only }	1.8	open
CapUpdateStatistics:	<i>boolean</i>	{ read-only }	1.8	open
CheckHealthText:	<i>string</i>	{ read-only }	1.0	open
Claimed:	<i>boolean</i>	{ read-only }	1.0	open
DataCount:	<i>int32</i>	{ read-only }	1.2	open
DataEventEnabled:	<i>boolean</i>	{ read-write }	1.0	open
DeviceEnabled:	<i>boolean</i>	{ read-write }	1.0	open & claim
FreezeEvents:	<i>boolean</i>	{ read-write }	1.0	open
OutputID:	<i>int32</i>	{ read-only }	1.0	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 }	1.0	--
DeviceControlDescription:	<i>string</i>	{ read-only }	1.0	--
DeviceControlVersion:	<i>int32</i>	{ read-only }	1.0	--
DeviceServiceDescription:	<i>string</i>	{ read-only }	1.0	open
DeviceServiceVersion:	<i>int32</i>	{ read-only }	1.0	open
PhysicalDeviceDescription:	<i>string</i>	{ read-only }	1.0	open
PhysicalDeviceName:	<i>string</i>	{ read-only }	1.0	open

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 }	1.0	open
CapJISOne:	<i>boolean</i>	{ read-only }	1.0	open
CapJISTwo:	<i>boolean</i>	{ read-only }	1.0	open
CapTransmitSentinels:	<i>boolean</i>	{ read-only }	1.5	open
AccountNumber:	<i>string</i>	{ read-only }	1.0	open
DecodeData:	<i>boolean</i>	{ read-write }	1.0	open
ErrorReportingType:	<i>int32</i>	{ read-write }	1.2	open
ExpirationDate:	<i>string</i>	{ read-only }	1.0	open
FirstName:	<i>string</i>	{ read-only }	1.0	open
MiddleInitial:	<i>string</i>	{ read-only }	1.0	open
ParseDecodeData:	<i>boolean</i>	{ read-write }	1.0	open
ServiceCode:	<i>string</i>	{ read-only }	1.0	open
Suffix:	<i>string</i>	{ read-only }	1.0	open
Surname:	<i>string</i>	{ read-only }	1.0	open
Title:	<i>string</i>	{ read-only }	1.0	open
Track1Data:	<i>binary</i>	{ read-only }	1.0	open
Track1DiscretionaryData:	<i>binary</i>	{ read-only }	1.0	open
Track2Data:	<i>binary</i>	{ read-only }	1.0	open
Track2DiscretionaryData:	<i>binary</i>	{ read-only }	1.0	open
Track3Data:	<i>binary</i>	{ read-only }	1.0	open
Track4Data:	<i>binary</i>	{ read-only }	1.5	open
TracksToRead:	<i>int32</i>	{ read-write }	1.0	open
TransmitSentinels:	<i>boolean</i>	{ read-write }	1.5	open

Methods (UML operations)**Common**

<i>Name</i>	<i>Version</i>
open (logicalDeviceName: <i>string</i>): void { raises-exception }	1.0
close (): void { raises-exception, use after open }	1.0
claim (timeout: <i>int32</i>): void { raises-exception, use after open }	1.0
release (): void { raises-exception, use after open, claim }	1.0
checkHealth (level: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.0
clearInput (): void { raises-exception, use after open, claim }	1.0
clearOutput (): void { }	<i>Not supported</i>
directIO (command: <i>int32</i>, inout data: <i>int32</i>, inout obj: <i>object</i>): void { raises-exception, use after open }	1.0
resetStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
retrieveStatistics (inout statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
updateStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>
upos::events::DataEvent			1.0
Status:	<i>int32</i>	{ read-only }	
upos::events::DirectIOEvent			1.0
EventNumber:	<i>int32</i>	{ read-only }	
Data:	<i>int32</i>	{ read-write }	
Obj:	<i>object</i>	{ read-write }	
upos::events::ErrorEvent			1.0
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		<i>Not supported</i>	
upos::events::StatusUpdateEvent			1.3
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:

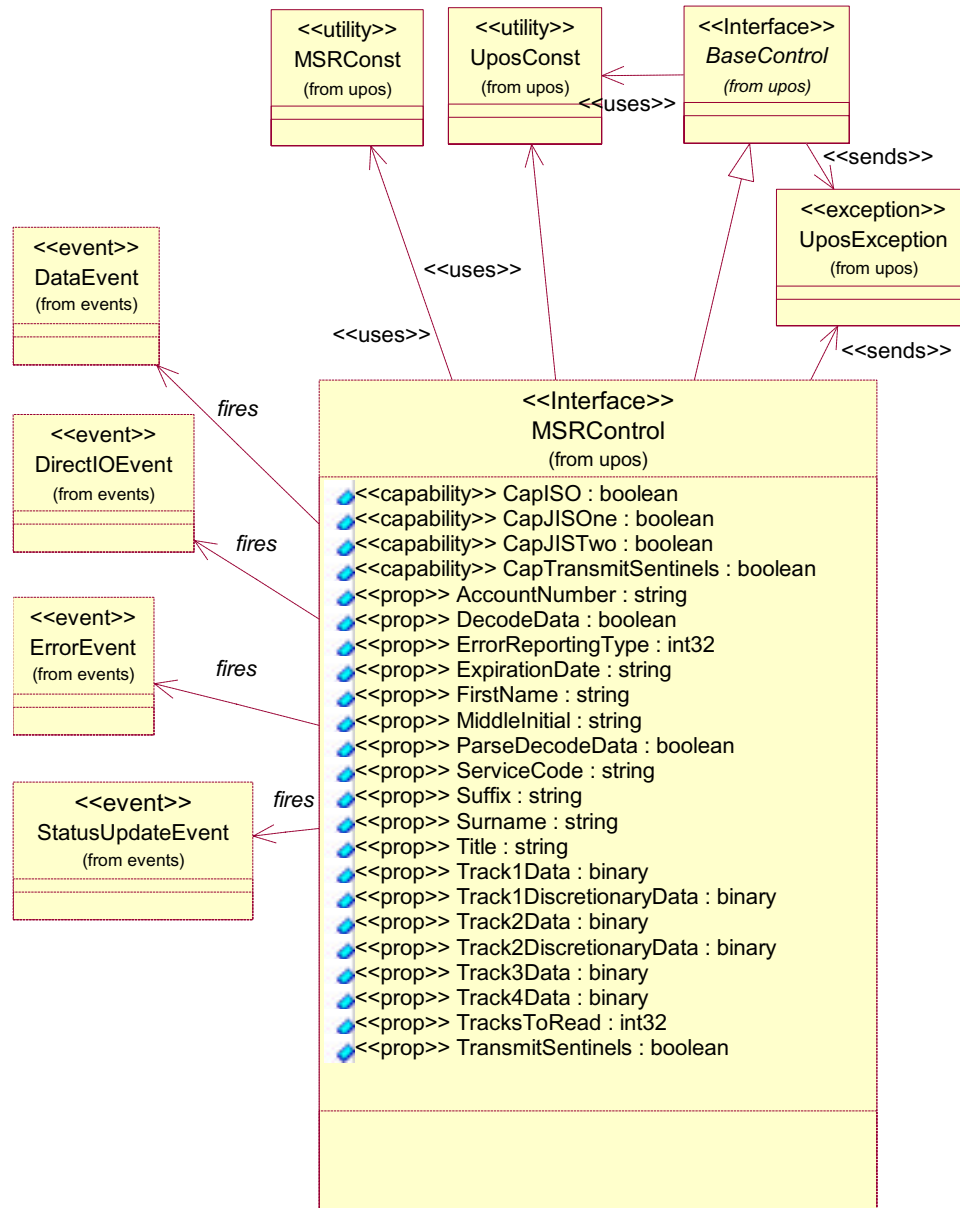
- Support for specific card types: ISO, JIS Type I and/or JIS Type II. Note: for the purpose of this standard, the following convention is assumed:
 - Track 1 is ISO or JIS-I Track 1
 - Track 2 is ISO or JIS-I Track 2
 - Track 3 is ISO or JIS-I Track 3
 - Track 4 is JIS-II data
 - Determination of the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

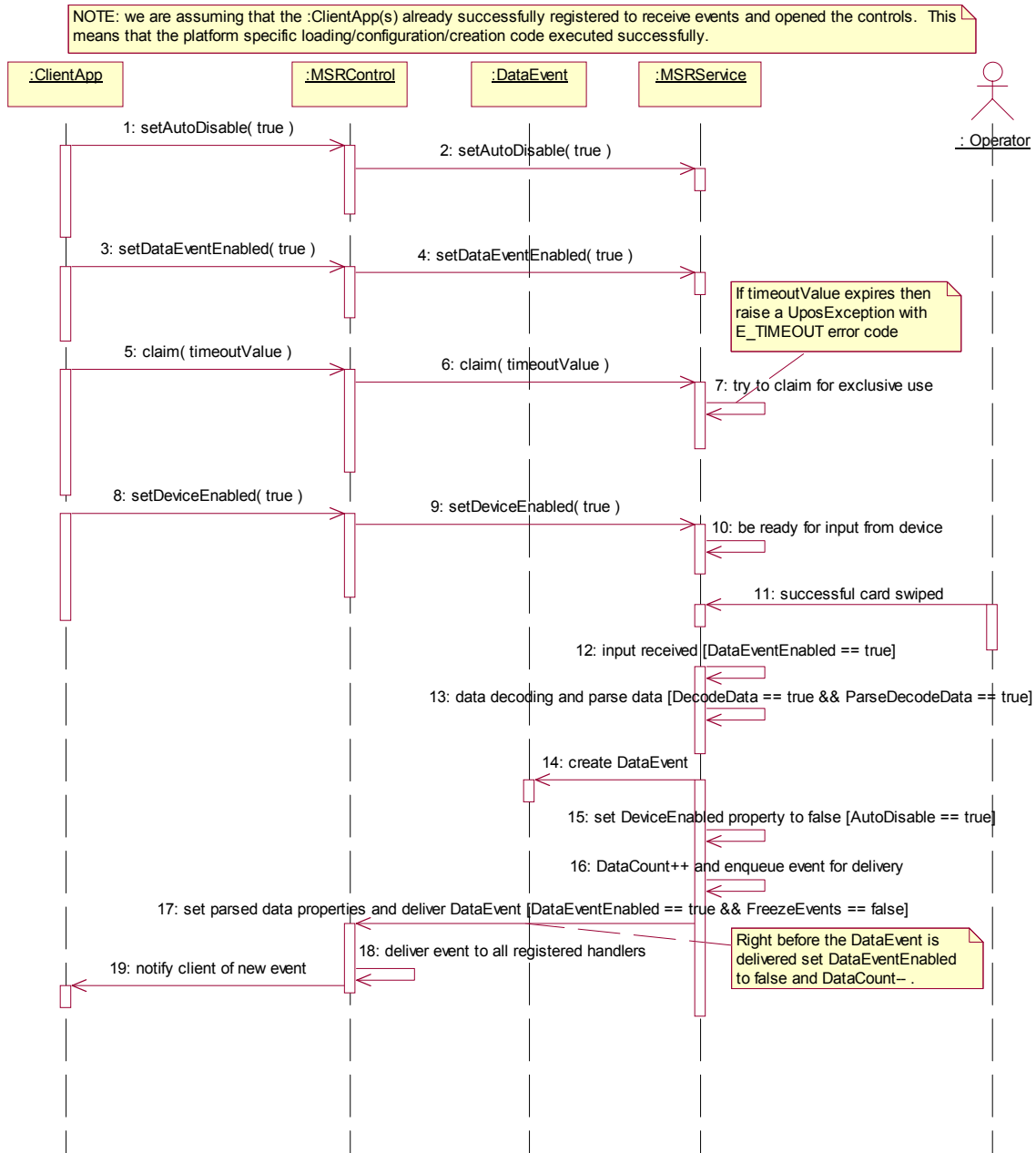
Device Sharing

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.

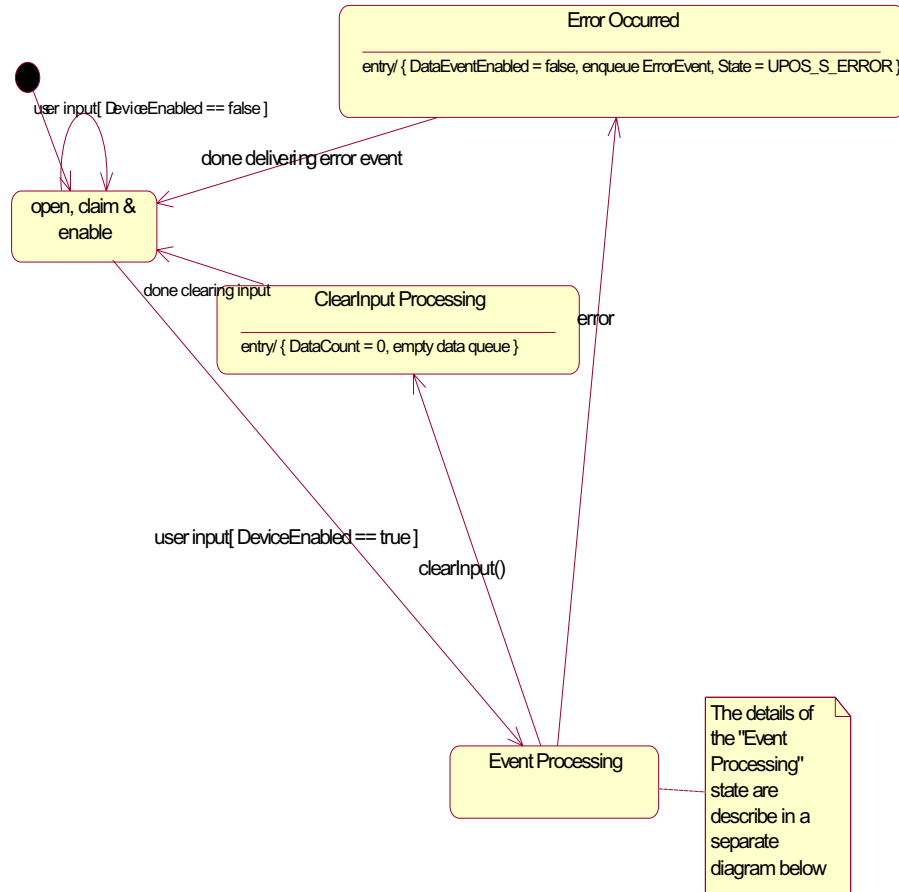
MSR Sequence Diagram Updated in Release 1.8

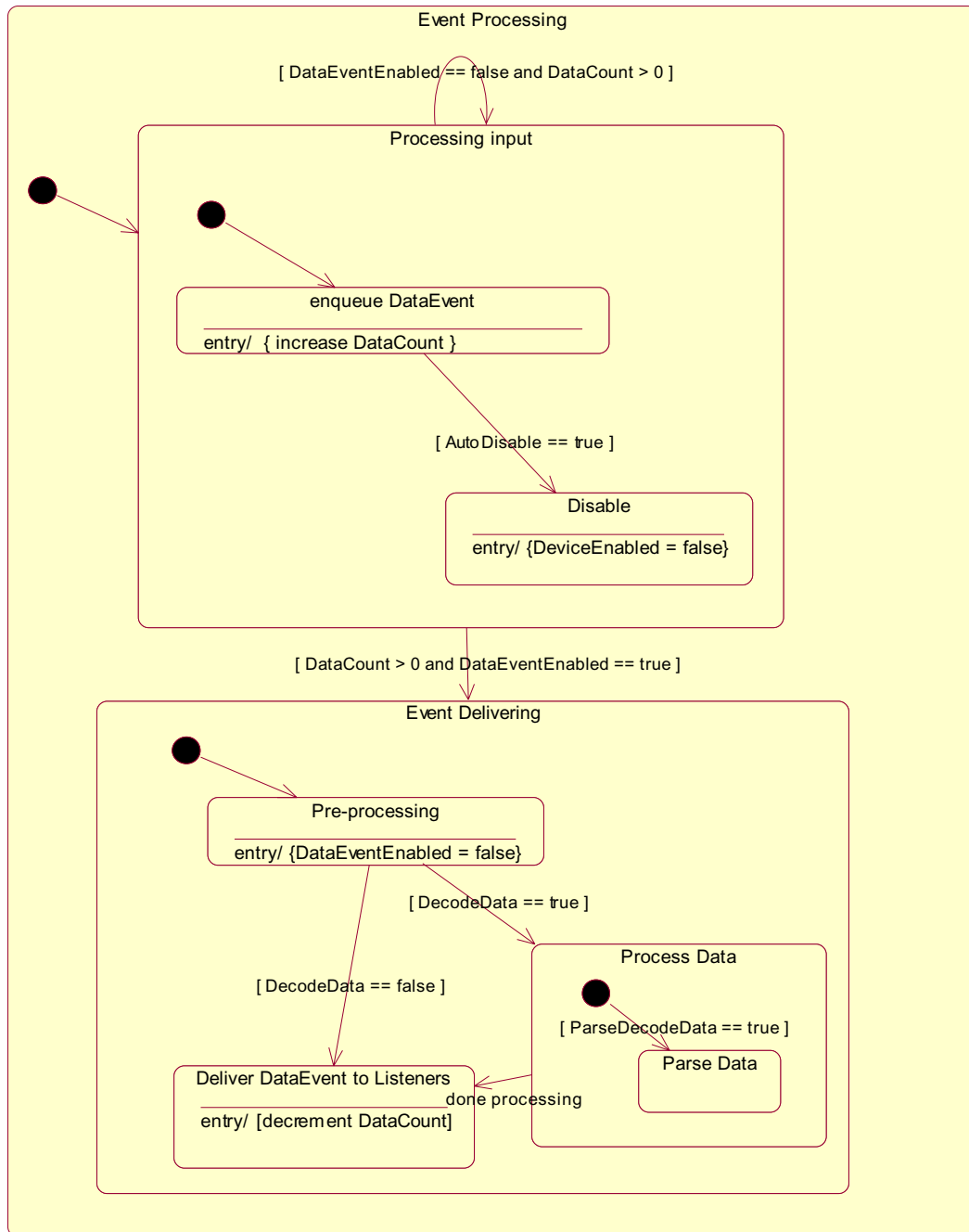
The following sequence diagram shows the typical usage of an MSR device.



MSR State Diagrams

The following state diagrams depict the MSR Control device model.





Properties (UML attributes)

AccountNumber Property

Syntax	AccountNumber: <i>string</i> { read-only, access after open }
Remarks	Holds the account number obtained from the most recently swiped card. This property is initialized to the empty string if: <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 16.
See Also	ParseDecodeData Property.

CapISO Property

Syntax	CapISO: <i>boolean</i> { read-only, access after open }
Remarks	If true, the MSR device supports ISO cards. 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 16.

CapJISOne Property

Syntax	CapJISOne: <i>boolean</i> { read-only, access after open }
Remarks	If true, the MSR device supports JIS Type-I cards. JIS-I cards are a superset of ISO cards. Therefore, if CapJISOne is true, then it is implied that CapISO is also true. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

CapJISTwo Property

Syntax	CapJISTwo: <i>boolean</i> { read-only, access after open }
Remarks	If true, the MSR device supports JIS type-II cards. 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 16.

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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
- 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
JIS-II	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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
- 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> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>MSR_ERT_CARD</td> <td>Report errors at a card level.</td> </tr> <tr> <td>MSR_ERT_TRACK</td> <td>Report errors at the track level</td> </tr> </tbody> </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.	MSR_ERT_TRACK	Report errors at the track level
Value	Meaning						
MSR_ERT_CARD	Report errors at a card level.						
MSR_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 16.						
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 16.
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 16.
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 16.
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 16.
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 16.
See Also	ParseDecodeData Property.

Suffix Property

Syntax	Suffix: <i>string</i> { read-only, access after open }
Remarks	Holds the suffix obtained from the most recently swiped card. This property is initialized to the empty string if: <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 16.
See Also	ParseDecodeData Property.

Surname Property

Syntax	Surname: <i>string</i> { read-only, access after open }
Remarks	Holds the surname obtained from the most recently swiped card. This property is initialized to the empty string if: <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 16.
See Also	ParseDecodeData Property.

Title Property

Syntax	Title: <i>string</i> { read-only, access after open }
Remarks	Holds the title obtained from the most recently swiped card. This property is initialized to the empty string if: <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 16.
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 16.
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 16.
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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
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 16.
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 16.
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 `UpoxException` may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.</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>The CapTransmitSentinels property is false.</td> </tr> </tbody> </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 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 MSR devices which may not have any knowledge of the Service's need for this event.

See Also "Events" on page 15, **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 attributes:

Attributes	Type	Description
<i>ErrorCode</i>	<i>int32</i>	Error code causing the error event. See a list of Error Codes on page 16.
<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 10 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 63.

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

See Also “Events” on page 15.

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</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
CapStatisticsReporting:	<i>boolean</i>	{ read-only }	1.8	open
CapUpdateStatistics:	<i>boolean</i>	{ read-only }	1.8	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

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>currency</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**

<i>Name</i>	<i>Version</i>
open (logicalDeviceName: <i>string</i>): void { raises-exception }	1.3
close (): void { raises-exception, use after open }	1.3
claim (timeout: <i>int32</i>): void { raises-exception, use after open }	1.3
release (): void { raises-exception, use after open, claim }	1.3
checkHealth (level: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3
clearInput (): void { raises-exception, use after open, claim, enable }	1.3
clearOutput (): void { }	<i>Not supported</i>
directIO (command: <i>int32</i>, inout data: <i>int32</i>, inout obj: <i>object</i>): void { raises-exception, use after open }	1.3
resetStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
retrieveStatistics (inout statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
updateStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8

Specific

<i>Name</i>	
beginEFTTransaction (PINPadSystem: <i>string</i>, transactionHost: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3
computeMAC (inMsg: <i>string</i>, outMsg: <i>object</i>): void { raises-exception, use after beginEFTTransaction }	1.3
enablePINEntry(): void { raises-exception, use after beginEFTTransaction }	1.3
endEFTTransaction (completionCode: <i>int32</i>): void { raises-exception, use after beginEFTTransaction }	1.3
updateKey (keyNum: <i>int32</i>, key: <i>string</i>): void { raises-exception, use after beginEFTTransaction }	1.3
verifyMAC (message: <i>string</i>): void { raises-exception, use after beginEFTTransaction }	1.3

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>
upos::events::DataEvent			1.3
Status:	<i>int32</i>	{ read-only }	
upos::events::DirectIOEvent			1.3
EventNumber:	<i>int32</i>	{ read-only }	
Data:	<i>int32</i>	{ read-write }	
Obj:	<i>object</i>	{ read-write }	
upos::events::ErrorEvent			1.3
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		<i>Not supported</i>	
upos::events::StatusUpdateEvent			1.3
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:

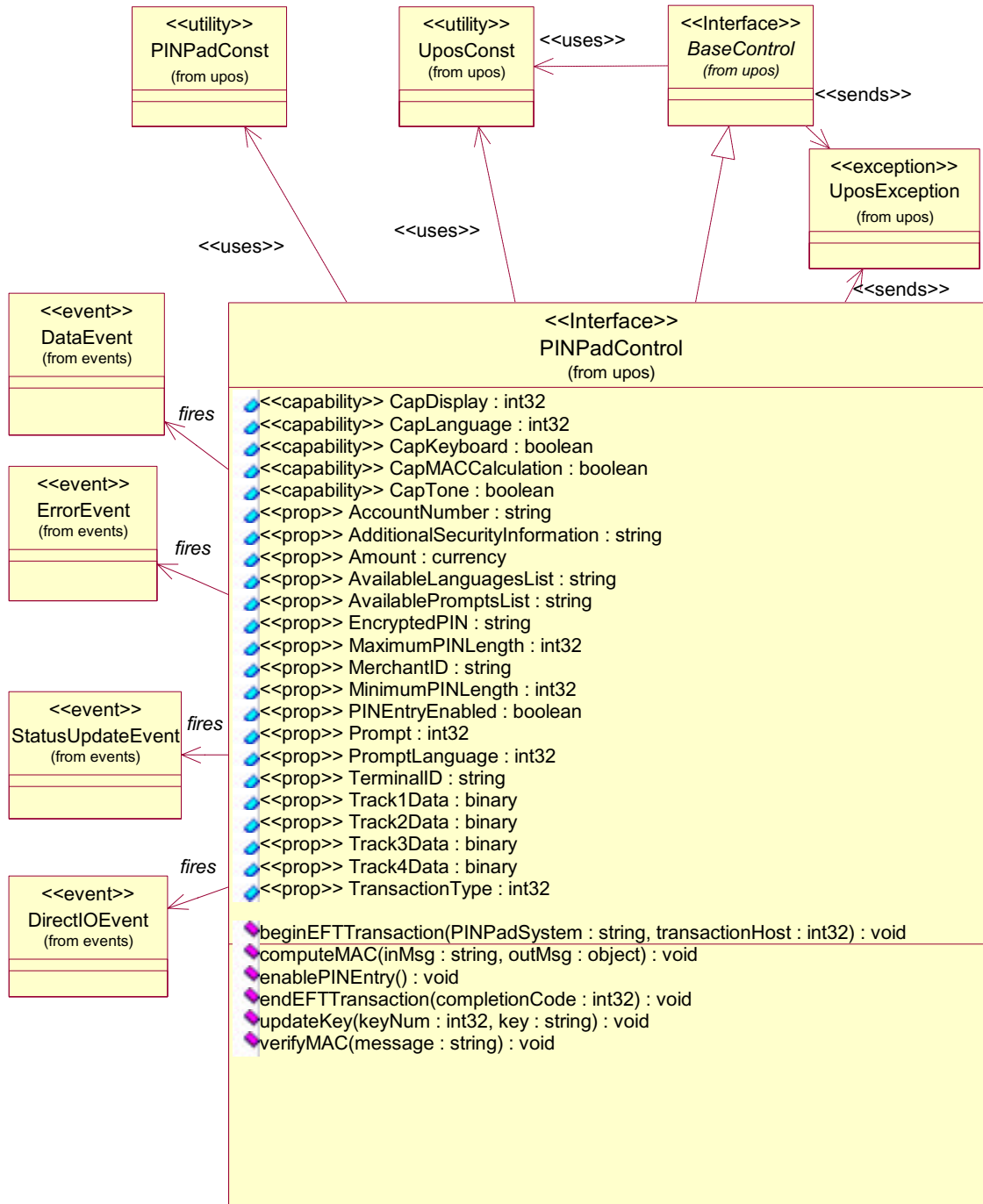
- Accept a PIN Entry at its keyboard and provide an Encrypted PIN to the application.

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, and 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:

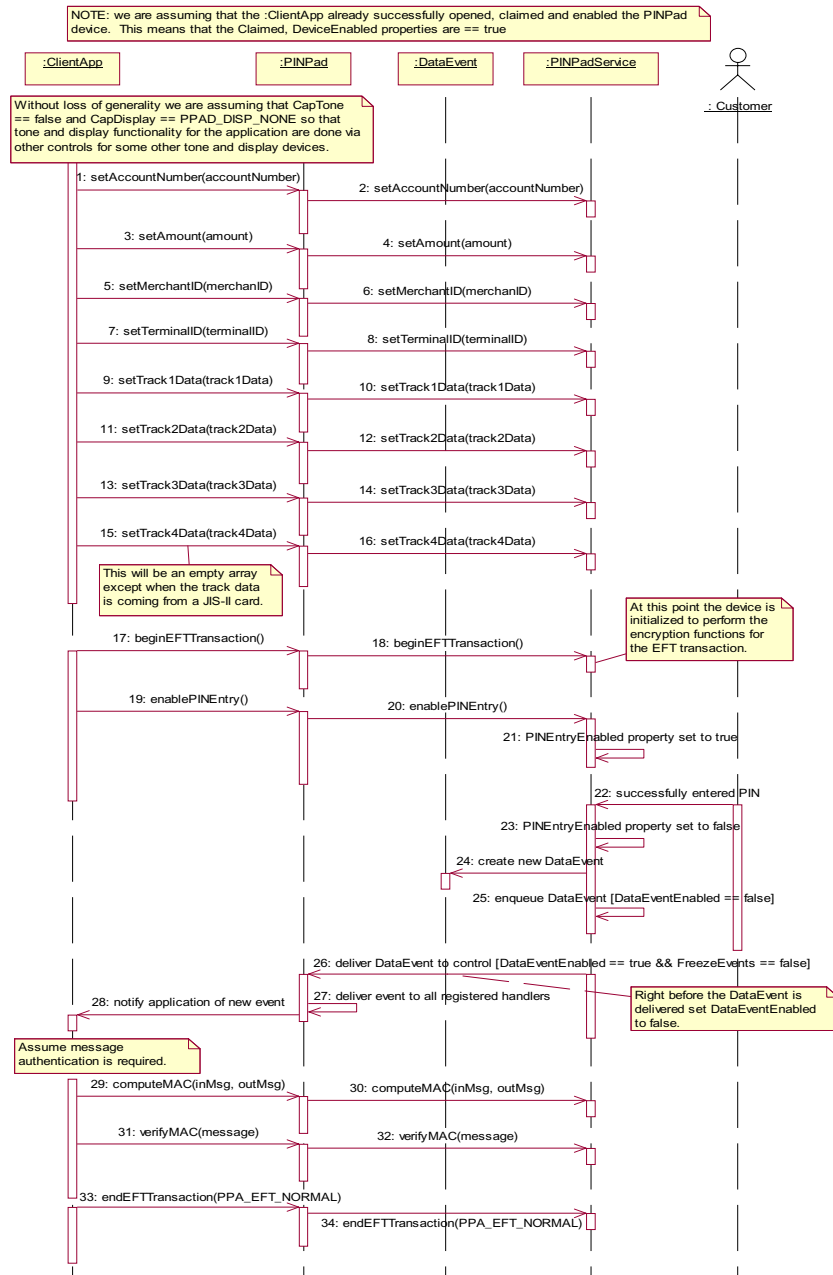
PIN Pad Class Diagram

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



PIN Pad Sequence Diagram *Added in Release 1.7*

The following sequence diagram shows the typical usage of a PIN Pad device, showing a general sequence of an application performing an EFT transaction with message authentication.



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 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.

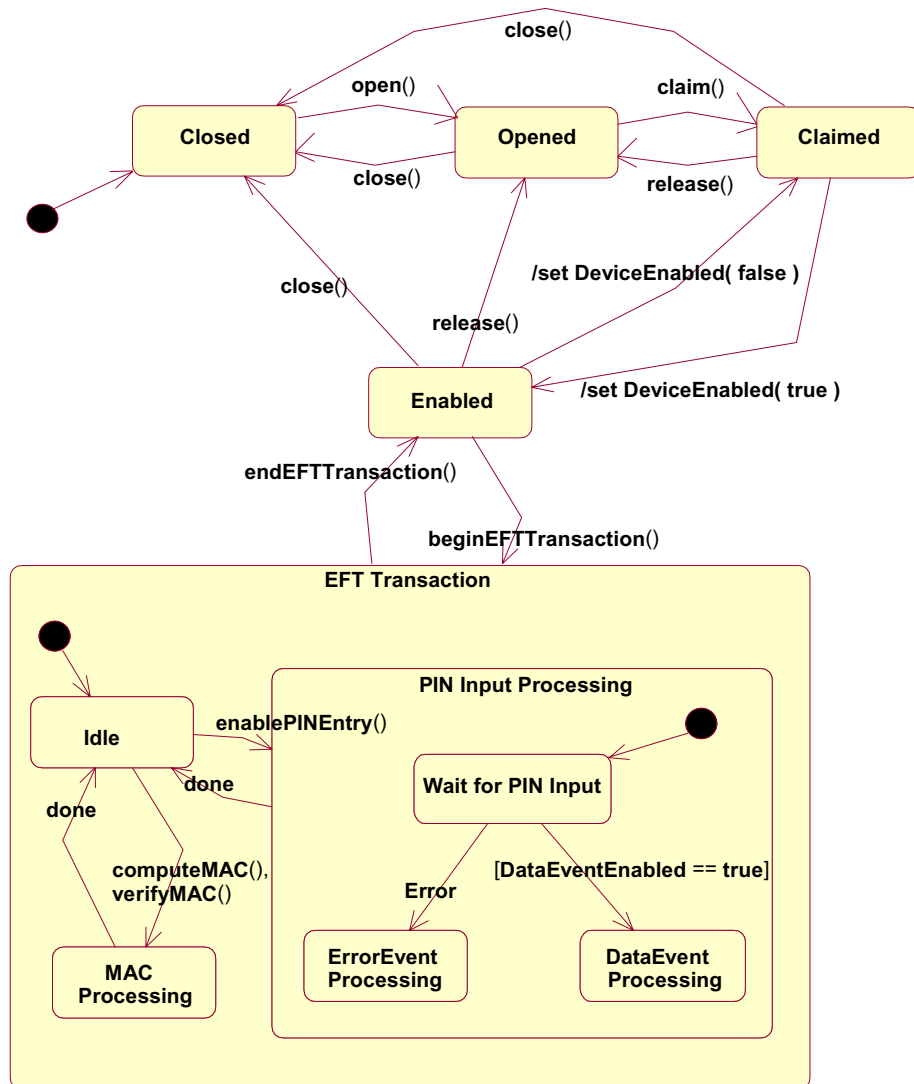
Device Sharing

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 16.

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.

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 16.

Amount Property

Corrected in Release 1.8

Syntax	Amount: <i>currency</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 16.

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.

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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
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 16.

See Also **Prompt** Property.

CapDisplay Property

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

Remarks Defines the operations that the application may perform on the PIN Pad display.

Value	Meaning
PPAD_DISP_UNRESTRICTED	The application can use the PIN Pad display in an unrestricted manner to display messages. In this case, an associated Line Display 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 16.

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 16.

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> <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 16.										
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 16.

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 16.

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 16.

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 16.

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 16.

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 16.

Prompt Property

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

Remarks 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.

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 16.

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

Value	Meaning
E_ILLEGAL	One of the following has occurred. * An attempt was made to set the property to a value that is not supported by the PIN Pad Service. * An attempt was made to select prompt messages in an unacceptable order (if CapDisplay is <code>PPAD_DISP_RESTRICTED_ORDER</code>).

See Also **PromptLanguage** Property.

PromptLanguage Property

- Syntax** **PromptLanguage:** *nls* { **read-write, access after open** }
- Remarks** 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**.
- A “language definition” consists of an ISO-639 language code and an ISO-3166 country code. The two codes are comma separated.
- The country code is optional and implies that the application does not care which country variant of the language is used.
- 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.
- The property is initialized to a default value by the **open** method.
- Errors** A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	One of the following occurred. * An attempt was made to set the property to a value that is not supported by the PIN Pad 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:** *string* { **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** A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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.

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	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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.

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	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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.

Track4Data Property**Added in Release 1.5****Syntax** **Track4Data: *binary* { read-write, access after open }****Remarks** 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.

To maintain compatibility with previous versions, the Control may also continue to store the JIS-II data in another **Track*n*Data** property. However, it should be noted that to ensure application portability, **Track4Data** should be used to access JIS-II data.

Errors A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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: *int32* { read-write, access after open }****Remarks** Holds the type of the current EFT Transaction. The application must set this property before calling the **beginEFTTransaction** method.

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 A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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.

computeMAC Method**Updated in Release 1.7**

Syntax	computeMAC (inMsg: <i>string</i>, outMsg: <i>object</i>): void { raises-exception, use after beginEFTTransaction)	
	Value	Description
	<i>inMsg</i> ¹	The message that the application intends to send to an EFT Transaction.
	<i>outMsg</i> ¹	Contains the result of applying the MAC calculation to <i>inMsg</i> . This output parameter will contain a reformatted message that may actually be transmitted to an EFT Transaction Host.
Remarks	Computes a MAC value and appends it to the designated message. Depending on the selected PIN Pad management system, the PIN Pad may also insert other fields into the message. Note that this method cannot be used while PIN Pad input (PIN Entry) is enabled.	
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.	
	Some possible values of the exception’s <i>ErrorCode</i> 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.

¹. In the OPOS environment, the format of this data depends upon the value of the **BinaryConversion** property. See **BinaryConversion** property on page A-28.

enablePINEntry Method

Syntax	enablePINEntry (): void { raises-exception, use after beginEFTTransaction);				
Remarks	<p>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.</p> <p>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.</p>				
Errors	<p>A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.</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_DISABLED</td> <td>A beginEFTTransaction method has not been performed.</td> </tr> </tbody> </table>	Value	Meaning	E_DISABLED	A beginEFTTransaction method has not been performed.
Value	Meaning				
E_DISABLED	A beginEFTTransaction method has not been performed.				

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 18.

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

See Also "Events" on page 15, **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 was detected while trying to perform a PIN encryption function.

Attributes This event contains the following attributes:

Attributes	Type	Description
<i>ErrorCode</i>	<i>int32</i>	Error code causing the error event. See a list of Error Codes on page 16.
<i>ErrorCodeExtended</i>	<i>int32</i>	Extended Error code causing the error event. If <i>ErrorCode</i> is <code>E_EXTENDED</code> , 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 <code>EL_INPUT</code> 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 processing may change *ErrorResponse* to the following value:

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 10 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>	Reports a change in the power state of a PIN Pad.

Note that Release 1.3 added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See "StatusUpdateEvent" description on page 63.

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

See Also "Events" on page 15.

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</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
CapStatisticsReporting:	<i>boolean</i>	{ read-only }	1.8	open
CapUpdateStatistics:	<i>boolean</i>	{ read-only }	1.8	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

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
CapMapCharacterSet:	<i>boolean</i>	{ read-only }	1.7	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
MapCharacterSet:	<i>boolean</i>	{ read-write }	1.7	open
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**

<i>Name</i>	<i>Version</i>
open (logicalDeviceName: <i>string</i>): void { raises-exception }	1.5
close (): void { raises-exception, use after open }	1.5
claim (timeout: <i>int32</i>): void { raises-exception, use after open }	1.5
release (): void { raises-exception, use after open, claim }	1.5
checkHealth (level: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.5
clearInput (): void { raises-exception, use after open, claim }	1.5
clearOutput (): void { raises-exception, use after open, claim }	1.5
directIO (command: <i>int32</i>, inout data: <i>int32</i>, inout obj: <i>object</i>): void { raises-exception, use after open }	1.5
resetStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
retrieveStatistics (inout statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
updateStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8

Specific

<i>Name</i>	<i>Version</i>
beginInsertion (timeout: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.5
beginRemoval (timeout: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.5
cleanCard (): void { raises-exception, use after open, claim, enable }	1.5
clearPrintWrite (kind: <i>int32</i>, hposition: <i>int32</i>, vposition: <i>int32</i>, width: <i>int32</i>, height: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.5
endInsertion (): void { raises-exception, use after open, claim, enable }	1.5
endRemoval (): void { raises-exception, use after open, claim, enable }	1.5
printWrite (kind: <i>int32</i>, hposition: <i>int32</i>, vposition: <i>int32</i>, data: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.5
rotatePrint (rotation: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.5

validateData (data: string): 1.5
void { raises-exception, use after open, claim, enable }

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>
upos::events::DataEvent			1.5
Status:	<i>int32</i>	{ read-only }	
upos::events::DirectIOEvent			1.5
EventNumber:	<i>int32</i>	{ read-only }	
Data:	<i>int32</i>	{ read-write }	
Obj:	<i>object</i>	{ read-write }	
upos::events::ErrorEvent			1.5
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			1.5
OutputID:	<i>int32</i>	{ read-only }	
upos::events::StatusUpdateEvent			1.5
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.

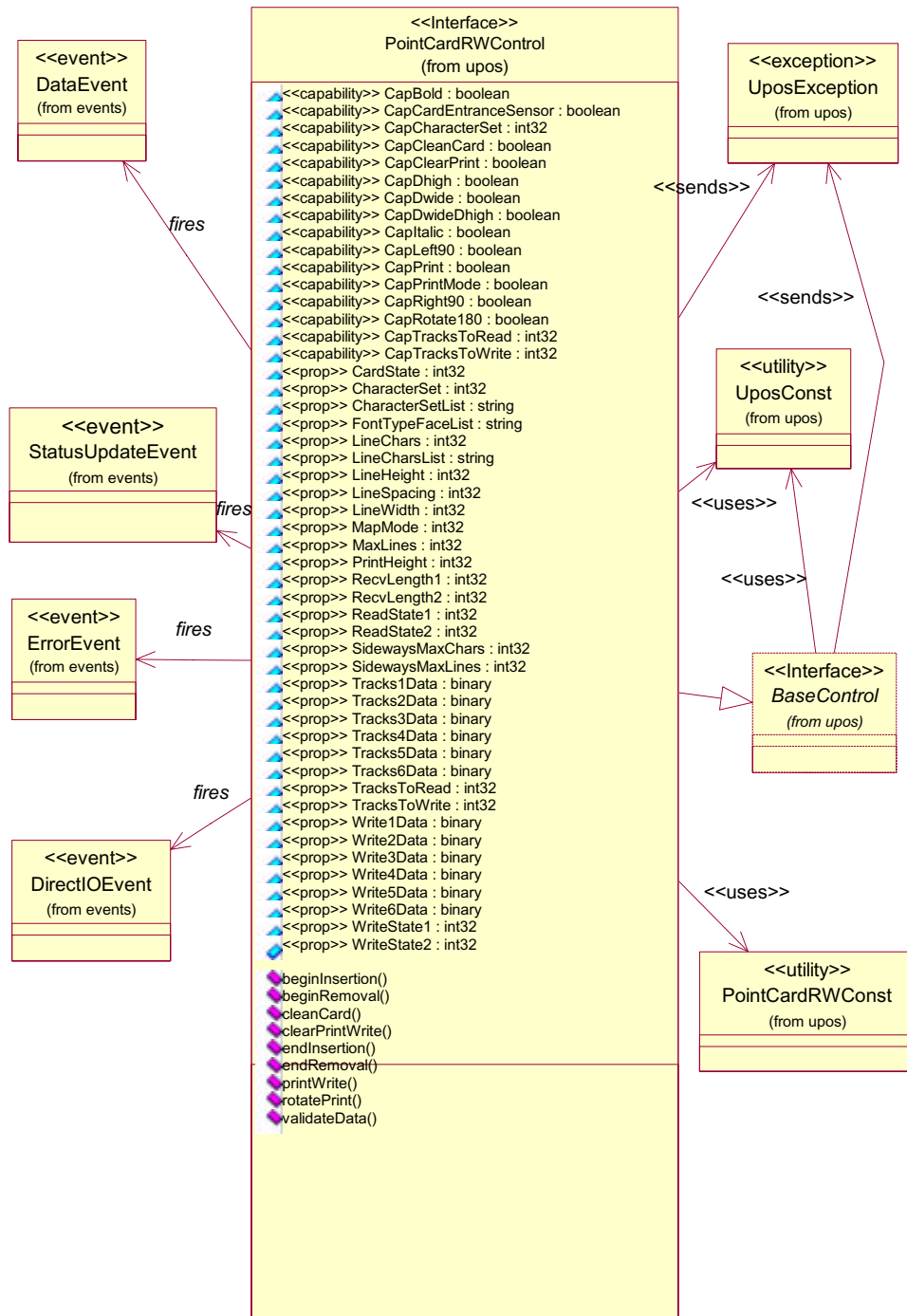
Capabilities

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.

Point Card Reader Writer Class Diagram

The following diagram shows the relationships between the Point Card Reader Writer classes.



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.

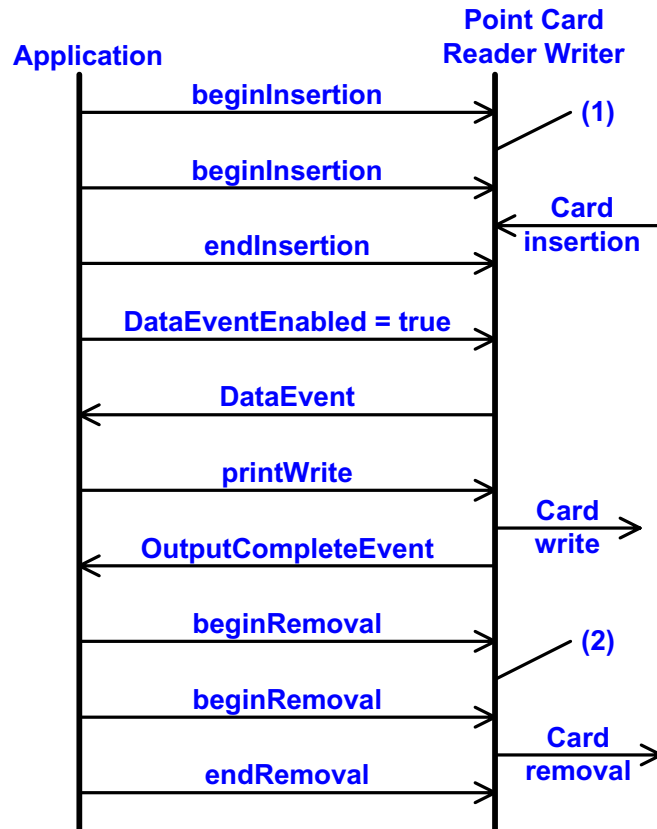
Output Model

Updated in Release 1.7

- 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 buffers the request in program memory, for delivery to the Physical Device as soon as the Physical Device can receive and process it, 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 buffered output data, including all asynchronous output, may be deleted by calling **clearOutput**. 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.

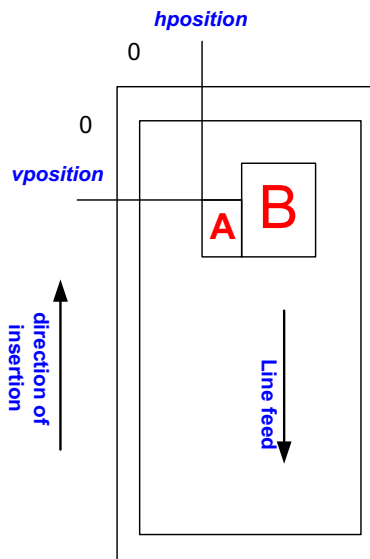


- (1) If the card is not inserted into the Point Card Reader Writer before the application specified timeout elapses, an exception is fired. The application needs to call **beginInsertion** again to confirm that a point card has been inserted or call **endInsertion** to cancel the card insertion. After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

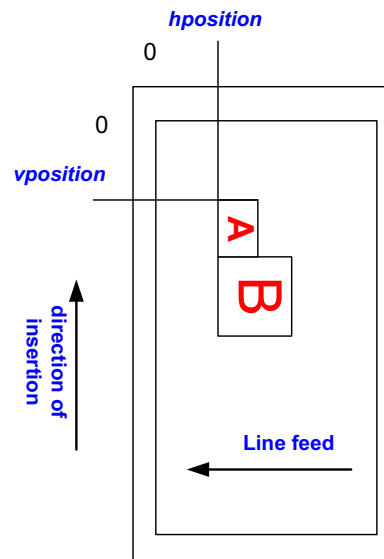
Printing Capability

- 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 *Updated in Release 1.7*

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.

If a sequence does not begin with ESC "|", or it begins with ESC "|", but is not a valid UnifiedPOS escape sequence, the Service will make a reasonable effort to pass it through to the Point Card Reader Writer. However, not all such sequences can be distinguished from printable data, so unexpected results may occur.

Starting with Release 1.7, the application can use the ESC|#E escape sequence to ensure more reliable handling of the amount of data to be passed through to the Point Card Reader Writer. Use of this escape sequence will make an application non-portable. The application may, however, maintain portability by performing Embedded Data Escape sequence calls within conditional code. This code may be based upon the value of the **DeviceServiceDescription**, the **PhysicalDeviceDescription**, or the **PhysicalDeviceName** property.

NOTE: This command sequence definition and the corresponding definition in the POS Printer Chapter, are the only known deviations from preserving the interchangeability of devices defined in this specification. If an application finds it necessary to utilize this command sequence, please inform the UnifiedPOS Committee (www.nrf-arts.org) with the details of its usage, so that a possible standard/generic Application Interface may be incorporated into a future release of the UnifiedPOS Standard. In order to preserve peripheral independence and interoperability at the Application level, it is the Committee's position that this command sequence should be used only as a "last resort".

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.

Commands Perform indicated action. *Added in Release 1.7*

Name	Data	Remarks
Pass through embedded data (See ^a below.)	ESC #E	Send the following # characters of data through to the hardware without modifying it. The character '#' is replaced by an ASCII decimal string telling the number of bytes following the escape sequence that should be passed through as-is to the hardware.

a. This escape sequence is only available in Version 1.7 and later.

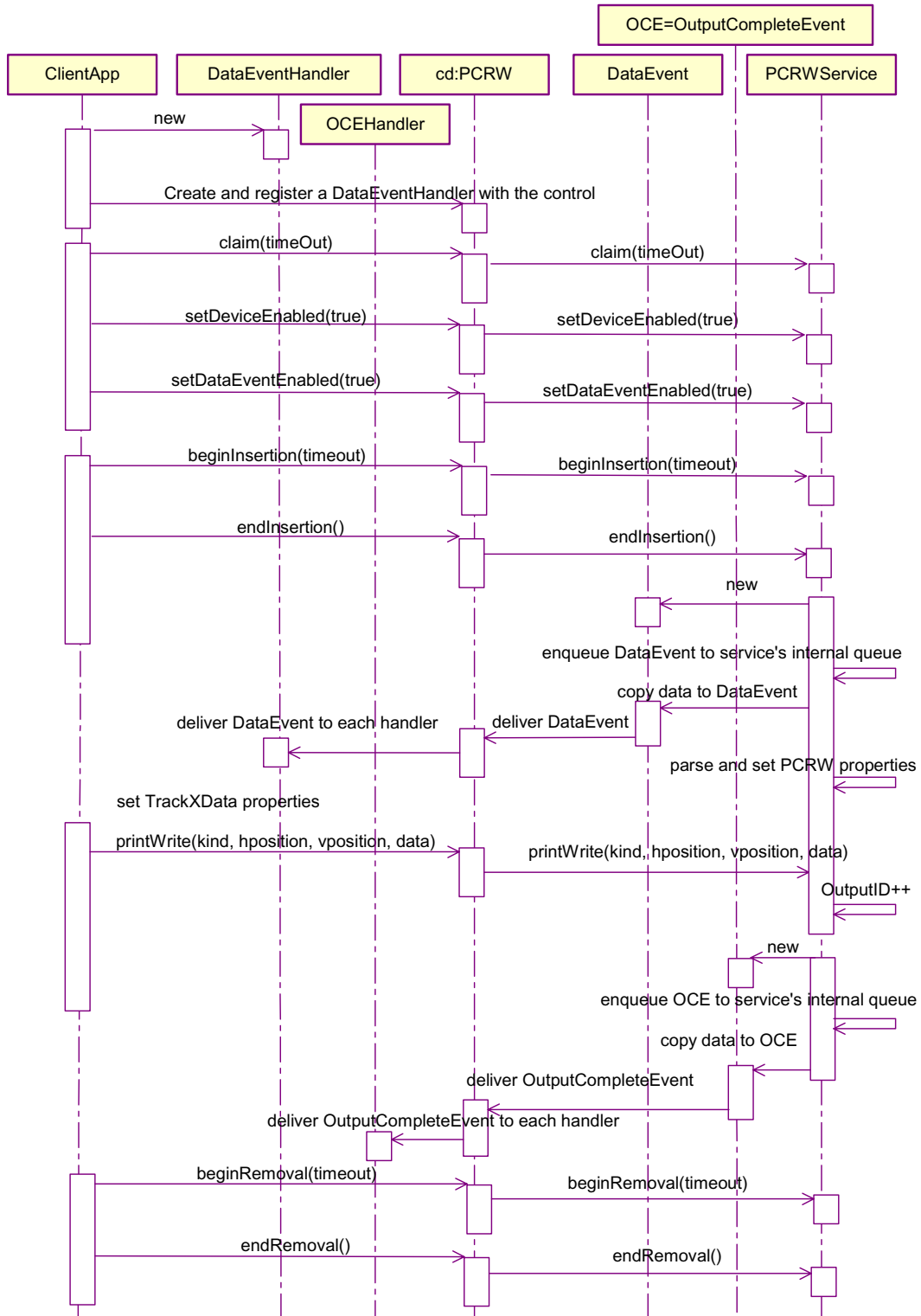
Print Mode Characteristics that are remembered until explicitly changed.

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

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 and wide	ESC 1C	Prints normal size.
Double wide	ESC 2C	Prints double-wide characters.
Double high	ESC 3C	Prints double-high characters.
Double high and 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 Sequence Diagram *Added in Release 1.7*



Properties (UML Attributes)

CapBold Property

Syntax	CapBold: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Point Card Reader Writer can print bold 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 16.

CapCardEntranceSensor Property

Syntax	CapCardEntranceSensor: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Point Card Reader Writer has an entrance sensor, 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 16.

CapCharacterSet Property

Syntax	CapCharacterSet: <i>int32</i> { read-only, access after open }												
Remarks	Holds the default character set capability. It may be one of the following: <table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>PCRW_CCS_ALPHA</td> <td>The default character set supports upper case alphabetic plus numeric, space, minus, and period.</td> </tr> <tr> <td>PCRW_CCS_ASCII</td> <td>The default character set supports all ASCII characters between 20-hex and 7F-hex.</td> </tr> <tr> <td>PCRW_CCS_KANA</td> <td>The default character set supports partial code page 932, including ASCII characters 20-hex through 7F-hex and the Japanese Kana characters A1-hex through DF-hex, but excluding the Japanese Kanji characters.</td> </tr> <tr> <td>PCRW_CCS_KANJI</td> <td>The default character set supports code page 932, including the Shift-JIS Kanji characters, Levels 1 and 2.</td> </tr> <tr> <td>PCRW_CCS_UNICODE</td> <td>The default character set supports UNICODE.</td> </tr> </tbody> </table> <p>The default character set may contain a superset of these ranges. The initial 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 16.												

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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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 16.

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 16.

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 16.

CapMapCharacterSet Property

Added in Release 1.7

Syntax	CapMapCharacterSet: <i>boolean</i> { read-only, access after open }
Remarks	Defines the ability of the Service to map the characters of the application to the selected character set when printing data. If CapMapCharacterSet is true, then the Service is able to map the characters to the character sets defined in CharacterSetList . 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 16.
See Also	CharacterSet Property, MapCharacterSet Property, CharacterSetList Property.

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 16.

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 16.

CapRight90 Property

Syntax	CapRight90: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Point Card Reader Writer can print in a rotated 90° right mode, false if it cannot. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

CapRotate180 Property

Syntax	CapRotate180: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Point Card Reader Writer can print in a rotated upside down mode, false if it cannot. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

CapTracksToRead Property

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

Remarks A bitmask indicating which magnetic tracks are accessible on the inserted point card. The value contained in this property is a bitwise OR of the constants PCRW_TRACK1 through PCRW_TRACK6.

For example, access to track 1 is possible when PCRW_TRACK1 is set.

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

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 16.

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 UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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 16.

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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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 #T). 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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	“Data Characters and Escape Sequences” on page 551.

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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.</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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

LineWidth Property

Syntax	LineWidth: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	The width of a line of LineChars characters. If CapPrintMode is true, expressed in the unit of measure given by MapMode . Setting LineChars may also update LineWidth . 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.
Errors	A UposException may be thrown when this property is accessed. For further information, see "Errors" on page 16.

MapCharacterSet Property

Added in Release 1.7

Syntax	MapCharacterSet: <i>boolean</i> { read-write, access after open }
Remarks	If MapCharacterSet is true and when outputting data, the Service maps the characters transferred by the application to the character set selected in the CharacterSet property for printing data. If MapCharacterSet is false, then no mapping is supported. In such a case the application has to ensure the mapping of the character set used in the application to the character set selected in the CharacterSet property. If CapMapCharacterSet is false, then this property is always 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 16.
See Also	CharacterSet Property, CapMapCharacterSet Property.

MapMode Property

Syntax	MapMode: <i>int32</i> { read-write, access after open-claim-enable }										
Remarks	Contains the mapping mode of the Point Card Reader Writer. The mapping mode defines the unit of measure used for other properties, such as line heights and line spacings. The following map modes are supported: <table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>PCRW_MM_DOTS</td> <td>The Point Card Reader Writer's dot width. This width may be different for each Point Card Reader Writer.</td> </tr> <tr> <td>PCRW_MM_TWIPS</td> <td>1/1440 of an inch.</td> </tr> <tr> <td>PCRW_MM_ENGLISH</td> <td>0.001 inch.</td> </tr> <tr> <td>PCRW_MM_METRIC</td> <td>0.01 millimeter.</td> </tr> </tbody> </table> 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.	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	A UposException may be thrown when this property is accessed. For further information, see "Errors" on page 16. Some possible values of the exception's <i>ErrorCode</i> property are: <table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>E_ILLEGAL</td> <td>An invalid mapping mode value was specified.</td> </tr> </tbody> </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 UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	LineHeight Property.

PrintHeight Property

Syntax	PrintHeight: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	<p>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.</p> <p>When the MapMode property is changed the value of the PrintHeight property changes.</p>
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
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 16.

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 16.

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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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 16.
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 16.
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 16.</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_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> </tbody> </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 16.</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_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> </tbody> </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	Contains the track 1 data from the point card. This property contains track data between but not including the start and end sentinels. An empty string indicates that the track was not accessible.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

Track2Data Property

Syntax	Track2Data: <i>binary</i> { read-only, access after open }
Remarks	Contains the track 2 data from the point card. This property contains track data between but not including the start and end sentinels. An empty string indicates that the track was not accessible.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

Track3Data Property

Syntax	Track3Data: <i>binary</i> { read-only, access after open }
Remarks	Contains the track 3 data from the point card. This property contains track data between but not including the start and end sentinels. An empty string indicates that the track was not accessible.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

Track4Data Property

Syntax	Track4Data: <i>binary</i> { read-only, access after open }
Remarks	Contains the track 4 data from the point card. This property contains track data between but not including the start and end sentinels. An empty string indicates that the track was not accessible.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

Track5Data Property

Syntax	Track5Data: <i>binary</i> { read-only, access after open }
Remarks	Contains the track 5 data from the point card. This property contains track data between but not including the start and end sentinels. An empty string indicates that the track was not accessible.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

Track6Data Property

Syntax	Track6Data: <i>binary</i> { read-only, access after open }
Remarks	Contains the track 6 data from the point card. This property contains track data between but not including the start and end sentinels. An empty string indicates that the track was not accessible.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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 16.

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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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 16.

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 16.

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 16.

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 16.

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 16.

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	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16. Some possible values of the exception’s <i>ErrorCode</i> property are:						
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>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 585.</td> </tr> </tbody> </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 585.
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 585.						
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 16.

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 585.

See Also **CapClearPrint** Property, **CapPrint** Property, **CapPrintMode** Property, **MapMode** Property.

endInsertion Method

Syntax	endInsertion (): void { raises-exception, use after open-claim-enable }								
Remarks	Called to end point card insertion processing. When called, the Point Card Reader Writer is taken out of point card insertion mode. If no point card is present, an exception is raised. This method is paired with the beginInsertion method for controlling point card insertion.								
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16. Some possible values of the exception’s <i>ErrorCode</i> property are:								
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>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 585.</td> </tr> </tbody> </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 585.
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 585.								
See Also	beginInsertion Method, beginRemoval Method, endRemoval Method.								

endRemoval Method

Syntax	endRemoval (): void { raises-exception, use after open-claim-enable }								
Remarks	Called to end point card removal processing. When called, the Point Card Reader Writer is taken out of point card removal or ejection mode. If a point card is present, an exception is raised. This method is paired with the beginRemoval method for controlling point card removal. The application may choose to call this method immediately after a successful beginRemoval if it wants to use the Point Card Reader Writer sensors to determine when the point card has been ejected. Alternatively, the application may prompt the user and wait for a key being pressed before calling this method.								
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16. Some possible values of the exception’s <i>ErrorCode</i> property are:								
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>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 585.</td> </tr> </tbody> </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 585.
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 585.								
See Also	beginInsertion Method, beginRemoval Method, endInsertion Method.								

printWrite Method**Updated in Release 1.7**

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> ¹	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 16.

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 585.

See Also **CapPrint** Property, **CapPrintMode** Property, **CapTracksToWrite** Property, **MapMode** Property, **TracksToWrite** Property, **WriteXData** Property.

¹. In the **OPOS** environment, the format of this data depends upon the value of the **BinaryConversion** property. See **BinaryConversion** property on page A-28.

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 15, **directIO** Method.

ErrorEvent**Updated in Release 1.7**

```

<< 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 attributes:

Attributes	Type	Description
<i>ErrorCode</i>	<i>int32</i>	Error code causing the error event. See a list of Error Codes on page 16.
<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 all buffered output data (including all 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 21.

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 status of the PointCard device.

The *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 63.

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 15, **CapCardEntranceSensor** Property.

POS Keyboard

This Chapter defines the POS Keyboard device category.

Summary

Properties (UML attributes)

<i>Common</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
AutoDisable:	<i>boolean</i>	{ read-write }	1.2	open
CapPowerReporting:	<i>int32</i>	{ read-only }	1.3	open
CapStatisticsReporting:	<i>boolean</i>	{ read-only }	1.8	open
CapUpdateStatistics:	<i>boolean</i>	{ read-only }	1.8	open
CheckHealthText:	<i>string</i>	{ read-only }	1.1	open
Claimed:	<i>boolean</i>	{ read-only }	1.1	open
DataCount:	<i>int32</i>	{ read-only }	1.2	open
DataEventEnabled:	<i>boolean</i>	{ read-write }	1.1	open
DeviceEnabled:	<i>boolean</i>	{ read-write }	1.1	open & claim
FreezeEvents:	<i>boolean</i>	{ read-write }	1.1	open
OutputID:	<i>int32</i>	{ read-only }	1.1	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 }	1.1	--
DeviceControlDescription:	<i>string</i>	{ read-only }	1.1	--
DeviceControlVersion:	<i>int32</i>	{ read-only }	1.1	--
DeviceServiceDescription:	<i>string</i>	{ read-only }	1.1	open
DeviceServiceVersion:	<i>int32</i>	{ read-only }	1.1	open
PhysicalDeviceDescription:	<i>string</i>	{ read-only }	1.1	open
PhysicalDeviceName:	<i>string</i>	{ read-only }	1.1	open

Properties (Continued)

<i>Specific</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
CapKeyUp:	<i>boolean</i>	{ read-only }	1.2	open
EventTypes:	<i>int32</i>	{ read-write }	1.2	open
POSKeyData:	<i>int32</i>	{ read-only }	1.1	open
POSKeyEventType:	<i>int32</i>	{ read-only }	1.2	open

Methods (UML operations)

Common

<i>Name</i>	<i>Version</i>
open (logicalDeviceName: <i>string</i>): void { raises-exception }	1.1
close (): void { raises-exception, use after open }	1.1
claim (timeout: <i>int32</i>): void { raises-exception, use after open }	1.1
release (): void { raises-exception, use after open, claim }	1.1
checkHealth (level: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.1
clearInput (): void { raises-exception, use after open, claim }	1.1
clearOutput (): void { }	<i>Not supported</i>
directIO (command: <i>int32</i> , inout data: <i>int32</i> , inout obj: <i>object</i>): void { raises-exception, use after open }	1.1
resetStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
retrieveStatistics (inout statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
updateStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8

Specific

None

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>
upos::events::DataEvent			1.1
Status:	<i>int32</i>	{ read-only }	
upos::events::DirectIOEvent			1.1
EventNumber:	<i>int32</i>	{ read-only }	
Data:	<i>int32</i>	{ read-write }	
Obj:	<i>object</i>	{ read-write }	
upos::events::ErrorEvent			1.1
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		<i>Not Supported</i>	
upos::events::StatusUpdateEvent			1.3
Status:	<i>int32</i>	{ read-only }	

General Information

The POS Keyboard programmatic name is “POSKeyboard”.

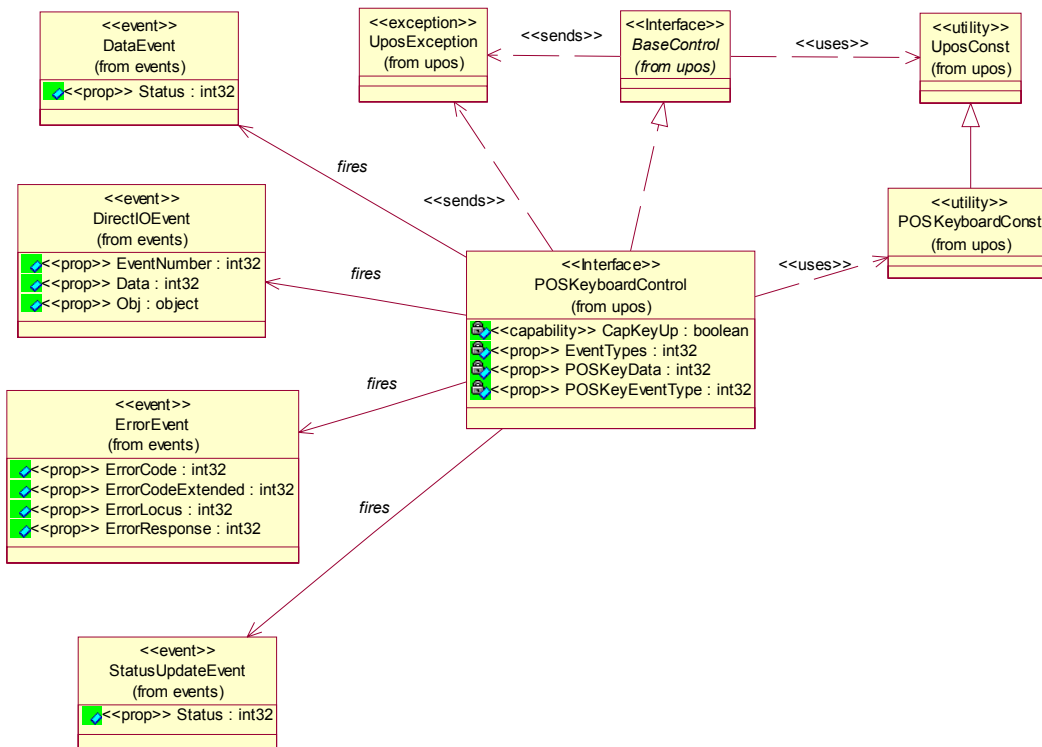
Capabilities

The POS Keyboard has the following capability:

- Reads keys from a POS keyboard. A POS keyboard may be an auxiliary keyboard, or it may be a virtual keyboard consisting of some or all of the keys on the system keyboard.

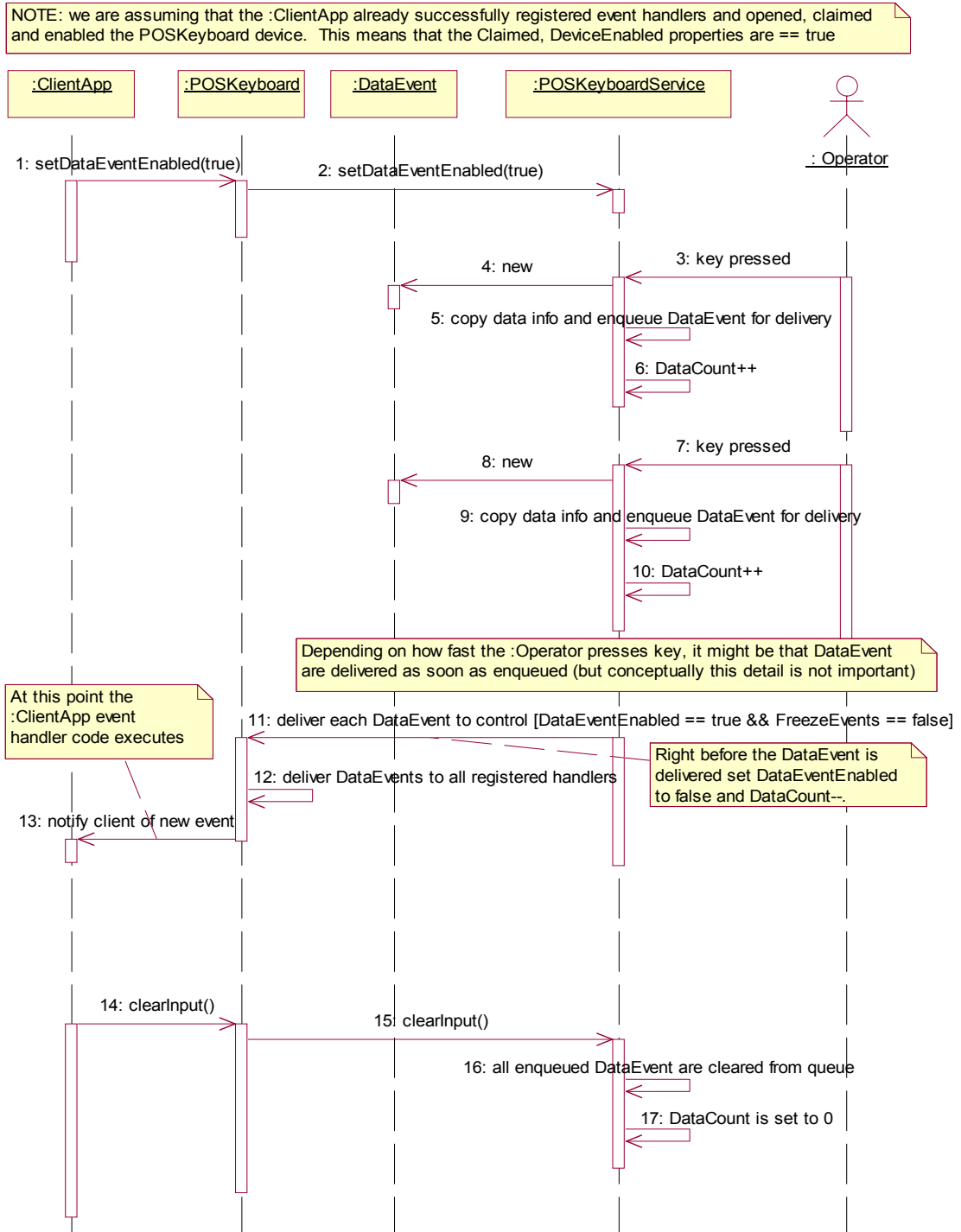
POS Keyboard Class Diagram

The following diagram shows the relationships between the POS Keyboard classes.



POS Keyboard Sequence Diagram *Updated in Release 1.8*

The following sequence diagram shows the typical usage of the POS Keyboard device.



Model

The POS Keyboard follows the general “Device Input Model” for input devices:

- When input is received from the POS Keyboard a **DataEvent** is enqueued.
- If the **AutoDisable** property is true, then the Device automatically disables itself when a **DataEvent** is enqueued.
- A queued **DataEvent** can be delivered to the application when the **DataEventEnabled** property is true and other event delivery requirements are met. Just before firing this event, data is copied into the properties, and further data events are disabled by setting **DataEventEnabled** to false. This causes subsequent input data to be enqueued while the application processes the current input and associated properties. When the application has finished the current input and is ready for more data, it reenables events by setting **DataEventEnabled** to true.
- An **ErrorEvent** (or events) is enqueued if an error occurs while gathering or processing input, and is delivered to the application when **DataEventEnabled** is true and other event delivery requirements are met.
- The **DataCount** property may be read to obtain the number of queued **DataEvents**.
- All queued input may be deleted by calling **clearInput**.

Keyboard Translation

The POS Keyboard Control must supply a mechanism for translating its internal key codes into user-defined codes which are returned by the **DataEvents**. Note that this translation *must* be end-user configurable.

Device Sharing

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

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

Properties (UML attributes)

CapKeyUp Property

Syntax	CapKeyUp: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the device is able to generate both key down and key up events, depending upon the setting of the EventTypes . If false, then the device is only able to generate the key down 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 16.
See Also	EventTypes Property.

EventTypes Property

Syntax	EventTypes: <i>int32</i> { read-write, access after open }						
Remarks	Holds the type of events that the application wants to receive. It has one of the following values: <table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>KBD_ET_DOWN</td> <td>Generate key down events.</td> </tr> <tr> <td>KBD_ET_DOWN_UP</td> <td>Generate key down and key up events.</td> </tr> </tbody> </table> <p>This property is initialized to KBD_ET_DOWN by the open method.</p>	Value	Meaning	KBD_ET_DOWN	Generate key down events.	KBD_ET_DOWN_UP	Generate key down and key up events.
Value	Meaning						
KBD_ET_DOWN	Generate key down events.						
KBD_ET_DOWN_UP	Generate key down and key up events.						
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.						

POSKeyData Property

Syntax	POSKeyData: <i>int32</i> { read-only, access after open }
Remarks	Holds the value of the key from the last DataEvent . The application may treat this value as device independent, assuming that the system installer has configured the Service to translate internal key codes to the codes expected by the application. Such configuration is inherently Service-specific. This property is set just before delivering the DataEvent .
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	DataEvent .

POSKeyEventProperty

Syntax	POSKeyEventProperty: <i>int32</i> { read-only, access after open }						
Remarks	<p>Holds the type of the last keyboard event: Is the key being pressed or released? It has one of the following values:</p> <table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>KBD_KET_KEYDOWN</td> <td>The key in POSKeyData was pressed.</td> </tr> <tr> <td>KBD_KET_KEYUP</td> <td>The key in POSKeyData was released.</td> </tr> </tbody> </table> <p>This property is set just before delivering the DataEvent.</p>	Value	Meaning	KBD_KET_KEYDOWN	The key in POSKeyData was pressed.	KBD_KET_KEYUP	The key in POSKeyData was released.
Value	Meaning						
KBD_KET_KEYDOWN	The key in POSKeyData was pressed.						
KBD_KET_KEYUP	The key in POSKeyData was released.						
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.						
See Also	POSKeyData Property, DataEvent .						

Events (UML interfaces)

DataEvent

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

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

Attributes This event contains the following attribute:

Attribute	Type	Description
<i>Status</i>	<i>int32</i>	Contains zero.

Remarks The logical key number is placed in the **POSKeyData** property and the event type is placed in the **POSKeyEventType** property before this event is delivered.

See Also **POSKeyData** Property, **POSKeyEventType** Property, “Events” on page 15

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 Keyboard 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 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 Keyboard devices which may not have any knowledge of the Service’s need for this event.

See Also “Events” on page 15, **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 was detected trying to read POS Keyboard data.

Attributes This event contains the following attributes:

Attribute	Type	Description
<i>ErrorCode</i>	<i>int32</i>	Error Code causing the error event. See list of <i>ErrorCodes</i> on page 16.
<i>ErrorCodeExtended</i>	<i>int32</i>	Extended Error Code causing the error event. 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 has one of the following values:

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 DataEventEnabled 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 POS Keyboard data. This event is not delivered until **DataEventEnabled** is true, so that proper application sequencing occurs.
- See Also** “Device Input Model” on page 18, “Device Information Reporting Model” on page 26

StatusUpdateEvent

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

Description Notifies the application when the working status of the POS Keyboard changes.

Attributes This event contains the following attribute:

Attribute	Type	Description
<i>Status</i>	<i>int32</i>	The status reported from the POS Keyboard.

Note that Release 1.3 added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See “StatusUpdateEvent” description on page 63.

- Remarks** Enqueued when the POS Keyboard needs to alert the application of a device state change.
- See Also** “Events” on page 15

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</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
CapStatisticsReporting:	<i>boolean</i>	{ read-only }	1.8	open
CapUpdateStatistics:	<i>boolean</i>	{ read-only }	1.8	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

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**

<i>Name</i>	<i>Version</i>
open (logicalDeviceName: <i>string</i>): void { raises-exception }	1.5
close (): void { raises-exception, use after open }	1.5
claim (timeout: <i>int32</i>): void { raises-exception, use after open }	1.5
release (): void { raises-exception, use after open, claim }	1.5
checkHealth (level: <i>int32</i>): void { raises-exception, use after open, enable }	1.5
clearInput (): void { }	<i>Not supported</i>
clearOutput (): void { }	<i>Not supported</i>
directIO (command: <i>int32</i>, inout data: <i>int32</i>, inout obj: <i>object</i>): void { raises-exception, use after open }	1.5
resetStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
retrieveStatistics (inout statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
updateStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8

Specific***Name***

shutdownPOS (): 1.5
 void { raises-exception, use after open, enable }

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>
upos::events::DataEvent		<i>Not Supported</i>	
upos::events::DirectIOEvent			1.5
EventNumber:	<i>int32</i>	{ read-only }	
Data:	<i>int32</i>	{ read-write }	
Obj:	<i>object</i>	{ read-write }	
upos::events::ErrorEvent		<i>Not Supported</i>	
upos::events::OutputCompleteEvent		<i>Not Supported</i>	
upos::events::StatusUpdateEvent			1.5
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 dependent 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.

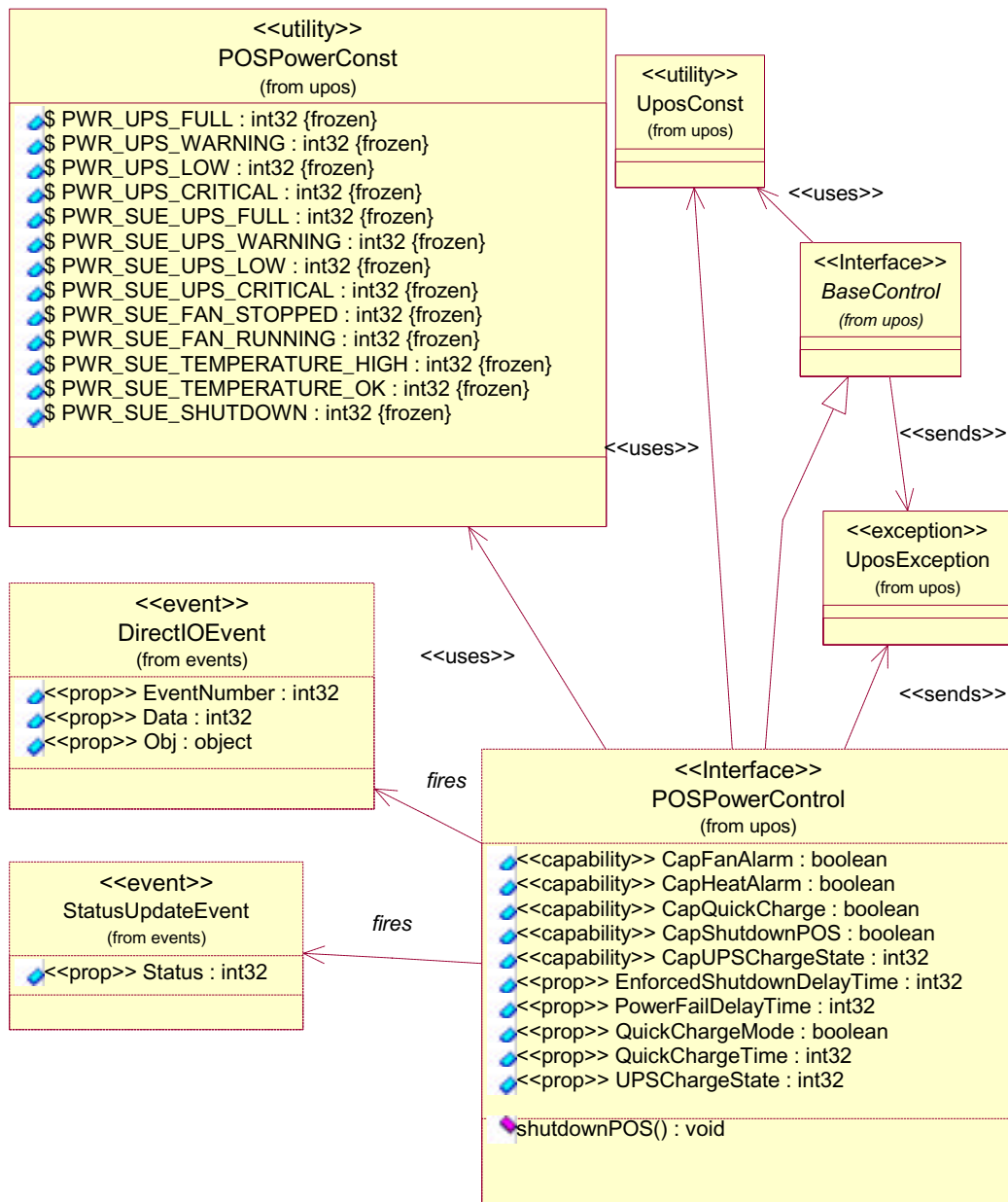
Power reporting only occurs while the device is open, enabled and power notification is switched on.

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.

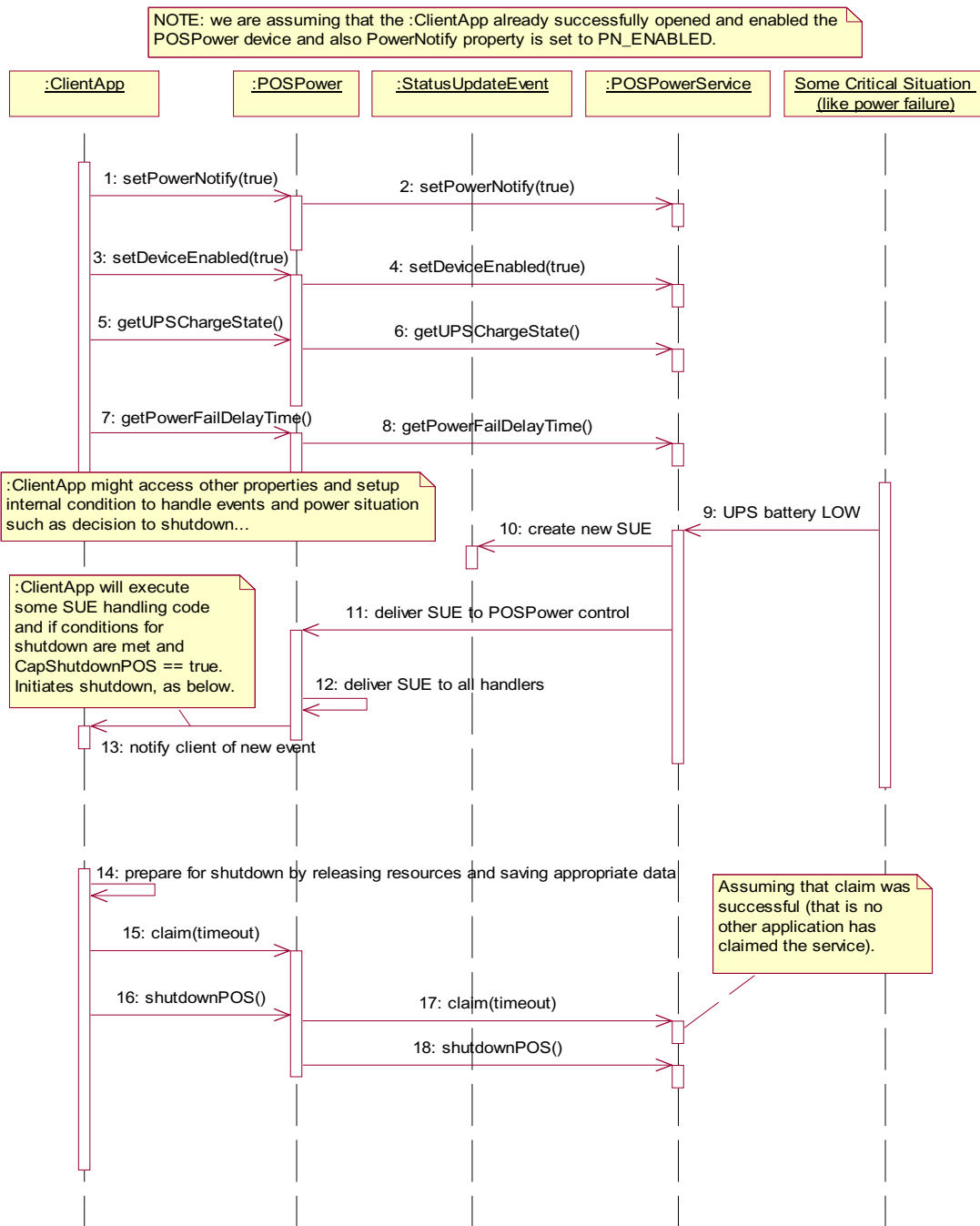
POSPower Class Diagram

The following diagram shows the relationships between the POSPower classes.



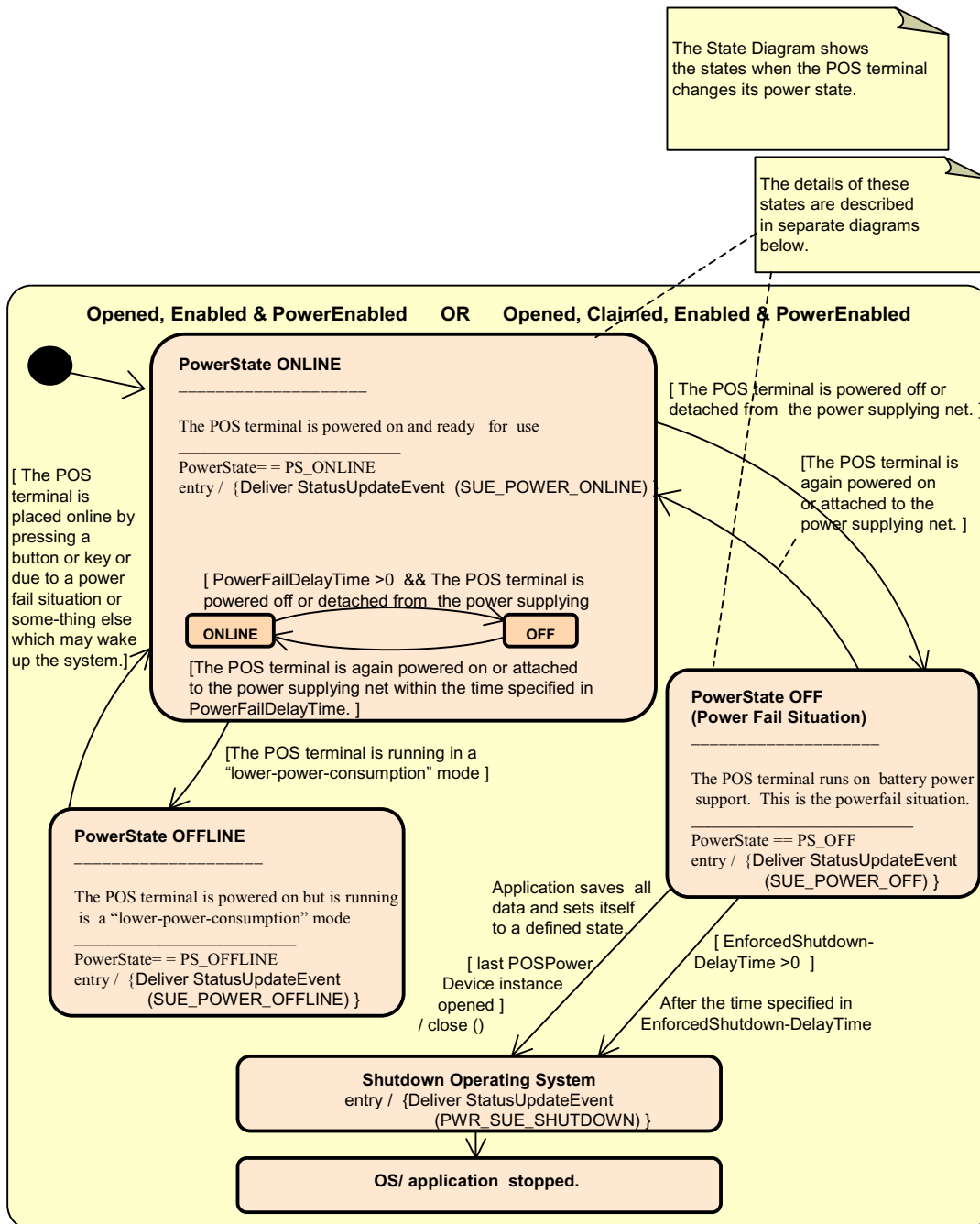
POSPower Sequence Diagram *Added in Release 1.7*

The following sequence diagram shows the typical usage of the POSPower device for registering for **StatusUpdateEvents** and an atypical case of initiating a **shutdownPOS** call.



POSPower PowerState Diagram - part 1

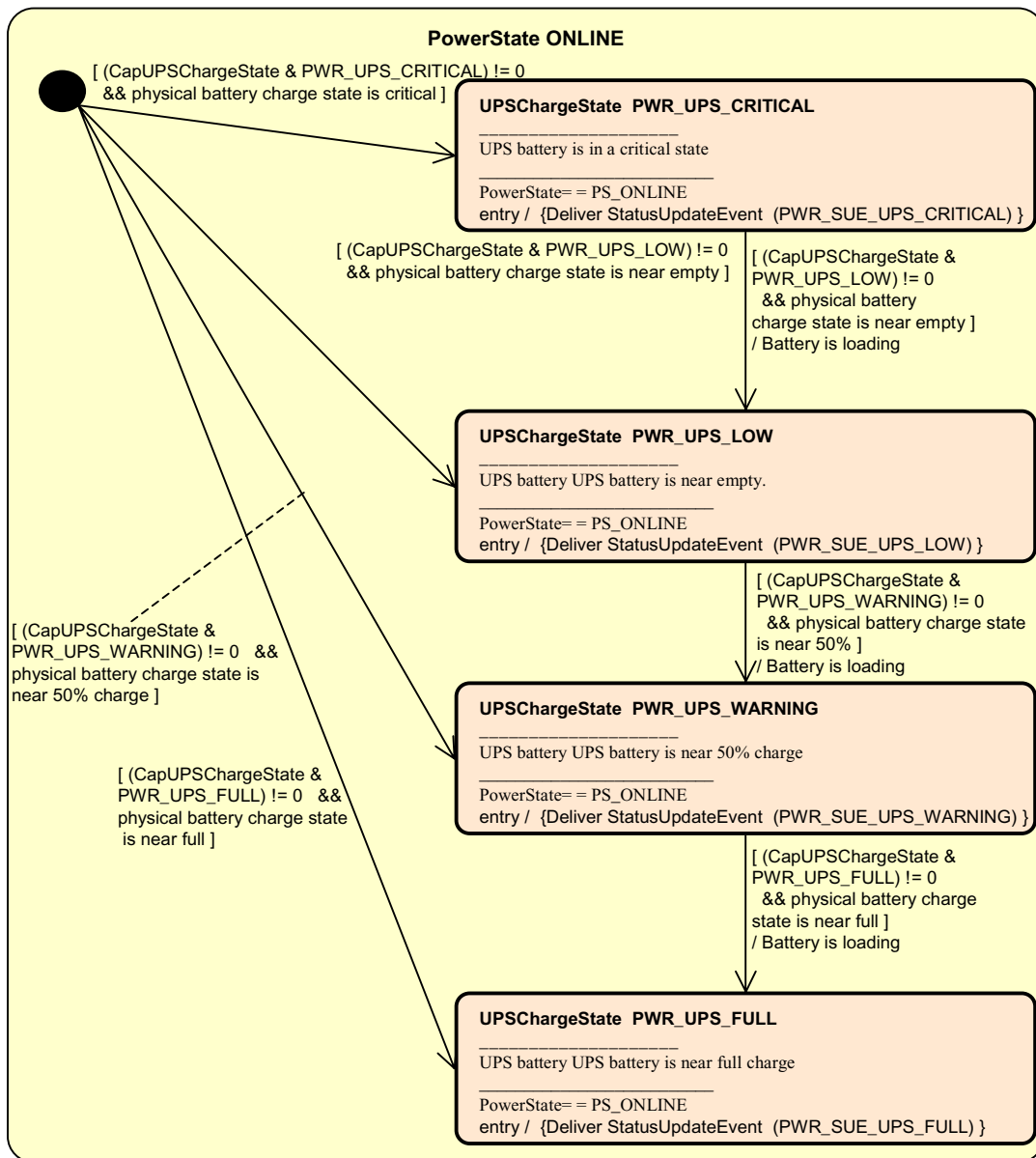
The following state diagram depicts the POSPower Power States.



POSPower PowerState Diagram - part 2

The following state diagram depicts the POSPower PowerState ONLINE.

The State Diagram shows the sub states in the PowerState ONLINE state when charging the UPS battery.



POSPower PowerState Diagram - part 3

The following state diagram depicts the POSPower PowerState OFF.

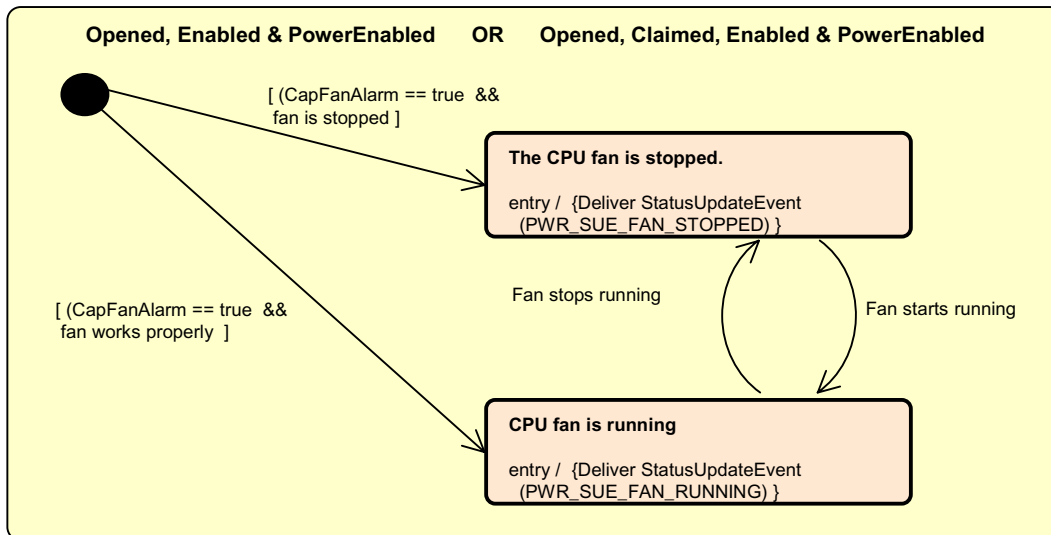
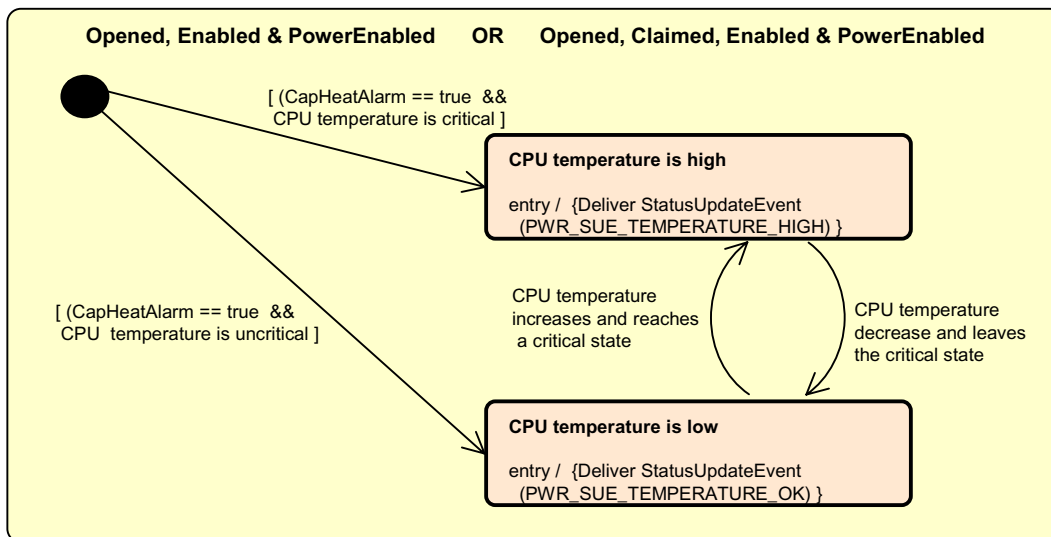
The State Diagram shows the sub states in the PowerState OFF state when unloading the UPS battery.



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 16.

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 16.

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 16.
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 16.
See Also	shutdownPOS Method.

CapUPSChargeState Property

Syntax	CapUPSChargeState: <i>int32</i> { read-only, access after open }										
Remarks	If not equal to zero, the UPS can deliver one or more charge states. It can contain any of the following values logically ORed together.										
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>PWR_UPS_FULL</td> <td>UPS battery is near full charge.</td> </tr> <tr> <td>PWR_UPS_WARNING</td> <td>UPS battery is near 50% charge.</td> </tr> <tr> <td>PWR_UPS_LOW</td> <td>UPS battery is near empty. Application shutdown should be started to ensure that it can be completed before the battery charge is depleted. A minimum of 2 minutes of normal system operation can be assumed when this state is entered unless this is the first state reported upon entering the “Off” power state.</td> </tr> <tr> <td>PWR_UPS_CRITICAL</td> <td>UPS battery is in a critical state and could be disconnected at any time without further warning.</td> </tr> </tbody> </table>	Value	Meaning	PWR_UPS_FULL	UPS battery is near full charge.	PWR_UPS_WARNING	UPS battery is near 50% charge.	PWR_UPS_LOW	UPS battery is near empty. Application shutdown should be started to ensure that it can be completed before the battery charge is depleted. A minimum of 2 minutes of normal system operation can be assumed when this state is entered unless this is the first state reported upon entering the “Off” power state.	PWR_UPS_CRITICAL	UPS battery is in a critical state and could be disconnected at any time without further warning.
Value	Meaning										
PWR_UPS_FULL	UPS battery is near full charge.										
PWR_UPS_WARNING	UPS battery is near 50% charge.										
PWR_UPS_LOW	UPS battery is near empty. Application shutdown should be started to ensure that it can be completed before the battery charge is depleted. A minimum of 2 minutes of normal system operation can be assumed when this state is entered unless this is the first state reported upon entering the “Off” power state.										
PWR_UPS_CRITICAL	UPS battery is in a critical state and could be disconnected at any time without further warning.										
	This property is initialized by the open method.										
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.										
See Also	UPSChargeState Property.										

EnforcedShutdownDelayTime Property

Syntax	EnforcedShutdownDelayTime: <i>int32</i> { read-write, access after open }
Remarks	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.
	Applications will be informed about an initiated automatic shutdown.
	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 16.
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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16

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. 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 16</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>This method is not supported (see the CapShutdownPOS property)</td> </tr> </tbody> </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 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 POSPower devices which may not have any knowledge of the Service's need for this event.

See Also "Events" on page 15, **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 *Status* property contains the updated power status or alarm status.

Value	Meaning
PWR_SUE_UPS_FULL	UPS battery is near full charge. Can be returned if 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 63.

See Also **CapFanAlarm** Property, **CapHeatAlarm** Property, **CapUPSChargeState** Property, **UPSChargeState** Property.

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</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
CapStatisticsReporting:	<i>boolean</i>	{ read-only }	1.8	open
CapUpdateStatistics:	<i>boolean</i>	{ read-only }	1.8	open
CheckHealthText:	<i>string</i>	{ read-only }	1.0	open
Claimed:	<i>boolean</i>	{ read-only }	1.0	open
DataCount:	<i>int32</i>	{ read-only }	1.2	Not Supported
DataEventEnabled:	<i>boolean</i>	{ read-write }	1.0	Not Supported
DeviceEnabled:	<i>boolean</i>	{ read-write }	1.0	open & claim
FreezeEvents:	<i>boolean</i>	{ read-write }	1.0	open
OutputID:	<i>int32</i>	{ read-only }	1.0	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 }	1.0	--
DeviceControlDescription:	<i>string</i>	{ read-only }	1.0	--
DeviceControlVersion:	<i>int32</i>	{ read-only }	1.0	--
DeviceServiceDescription:	<i>string</i>	{ read-only }	1.0	open
DeviceServiceVersion:	<i>int32</i>	{ read-only }	1.0	open
PhysicalDeviceDescription:	<i>string</i>	{ read-only }	1.0	open
PhysicalDeviceName:	<i>string</i>	{ read-only }	1.0	open

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 }	1.1	open
CapConcurrentJrnRec:	<i>boolean</i>	{ read-only }	1.0	open
CapConcurrentJrnSlp:	<i>boolean</i>	{ read-only }	1.0	open
CapConcurrentRecSlp:	<i>boolean</i>	{ read-only }	1.0	open
CapCoverSensor:	<i>boolean</i>	{ read-only }	1.0	open
CapMapCharacterSet:	<i>boolean</i>	{ read-only }	1.7	open
CapTransaction:	<i>boolean</i>	{ read-only }	1.1	open
CapJrnPresent:	<i>boolean</i>	{ read-only }	1.0	open
CapJrn2Color:	<i>boolean</i>	{ read-only }	1.0	open
CapJrnBold:	<i>boolean</i>	{ read-only }	1.0	open
CapJrnDhigh:	<i>boolean</i>	{ read-only }	1.0	open
CapJrnDwide:	<i>boolean</i>	{ read-only }	1.0	open
CapJrnDwideDhigh:	<i>boolean</i>	{ read-only }	1.0	open
CapJrnEmptySensor:	<i>boolean</i>	{ read-only }	1.0	open
CapJrnItalic:	<i>boolean</i>	{ read-only }	1.0	open
CapJrnNearEndSensor:	<i>boolean</i>	{ read-only }	1.0	open
CapJrnUnderline:	<i>boolean</i>	{ read-only }	1.0	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 }	1.0	open
CapRec2Color:	<i>boolean</i>	{ read-only }	1.0	open
CapRecBarCode:	<i>boolean</i>	{ read-only }	1.0	open
CapRecBitmap:	<i>boolean</i>	{ read-only }	1.0	open
CapRecBold:	<i>boolean</i>	{ read-only }	1.0	open
CapRecDhigh:	<i>boolean</i>	{ read-only }	1.0	open
CapRecDwide:	<i>boolean</i>	{ read-only }	1.0	open
CapRecDwideDhigh:	<i>boolean</i>	{ read-only }	1.0	open
CapRecEmptySensor:	<i>boolean</i>	{ read-only }	1.0	open
CapRecItalic:	<i>boolean</i>	{ read-only }	1.0	open
CapRecLeft90:	<i>boolean</i>	{ read-only }	1.0	open
CapRecNearEndSensor:	<i>boolean</i>	{ read-only }	1.0	open
CapRecPapercut:	<i>boolean</i>	{ read-only }	1.0	open
CapRecRight90:	<i>boolean</i>	{ read-only }	1.0	open
CapRecRotate180:	<i>boolean</i>	{ read-only }	1.0	open
CapRecStamp:	<i>boolean</i>	{ read-only }	1.0	open
CapRecUnderline:	<i>boolean</i>	{ read-only }	1.0	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 }	1.0	open
CapSlpFullslip:	<i>boolean</i>	{ read-only }	1.0	open
CapSlp2Color:	<i>boolean</i>	{ read-only }	1.0	open
CapSlpBarCode:	<i>boolean</i>	{ read-only }	1.0	open
CapSlpBitmap:	<i>boolean</i>	{ read-only }	1.0	open
CapSlpBold:	<i>boolean</i>	{ read-only }	1.0	open
CapSlpDhigh:	<i>boolean</i>	{ read-only }	1.0	open
CapSlpDwide:	<i>boolean</i>	{ read-only }	1.0	open
CapSlpDwideDhigh:	<i>boolean</i>	{ read-only }	1.0	open
CapSlpEmptySensor:	<i>boolean</i>	{ read-only }	1.0	open
CapSlpItalic:	<i>boolean</i>	{ read-only }	1.0	open
CapSlpLeft90:	<i>boolean</i>	{ read-only }	1.0	open
CapSlpNearEndSensor:	<i>boolean</i>	{ read-only }	1.0	open
CapSlpRight90:	<i>boolean</i>	{ read-only }	1.0	open
CapSlpRotate180:	<i>boolean</i>	{ read-only }	1.0	open
CapSlpUnderline:	<i>boolean</i>	{ read-only }	1.0	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 }	1.0	open
CartridgeNotify:	<i>int32</i>	{ read-write }	1.5	open
CharacterSet:	<i>int32</i>	{ read-write }	1.0	open, claim, & enable
CharacterSetList:	<i>string</i>	{ read-only }	1.0	open
CoverOpen:	<i>boolean</i>	{ read-only }	1.0	open, claim, & enable
ErrorLevel:	<i>int32</i>	{ read-only }	1.1	open
ErrorStation:	<i>int32</i>	{ read-only }	1.0	open
ErrorString:	<i>string</i>	{ read-only }	1.1	open
FontTypefaceList:	<i>string</i>	{ read-only }	1.1	open
FlagWhenIdle:	<i>boolean</i>	{ read-write }	1.0	open
MapCharacterSet:	<i>boolean</i>	{ read-write }	1.7	open
MapMode:	<i>int32</i>	{ read-write }	1.0	open
RotateSpecial:	<i>int32</i>	{ read-write }	1.1	open
JrnLineChars:	<i>int32</i>	{ read-write }	1.0	open, claim, & enable
JrnLineCharsList:	<i>string</i>	{ read-only }	1.0	open
JrnLineHeight:	<i>int32</i>	{ read-write }	1.0	open, claim, & enable
JrnLineSpacing:	<i>int32</i>	{ read-write }	1.0	open, claim, & enable
JrnLineWidth:	<i>int32</i>	{ read-only }	1.0	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 }	1.0	open, claim, & enable
JrnEmpty:	<i>boolean</i>	{ read-only }	1.0	open, claim, & enable
JrnNearEnd:	<i>boolean</i>	{ read-only }	1.0	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 }	1.0	open, claim, & enable
RecLineCharsList:	<i>string</i>	{ read-only }	1.0	open
RecLineHeight:	<i>int32</i>	{ read-write }	1.0	open, claim, & enable
RecLineSpacing:	<i>int32</i>	{ read-write }	1.0	open, claim, & enable
RecLineWidth:	<i>int32</i>	{ read-only }	1.0	open, claim, & enable
RecLetterQuality:	<i>boolean</i>	{ read-write }	1.0	open, claim, & enable
RecEmpty:	<i>boolean</i>	{ read-only }	1.0	open, claim, & enable
RecNearEnd:	<i>boolean</i>	{ read-only }	1.0	open, claim, & enable
RecSidewaysMaxLines:	<i>int32</i>	{ read-only }	1.0	open, claim, & enable
RecSidewaysMaxChars:	<i>int32</i>	{ read-only }	1.0	open, claim, & enable
RecLinesToPaperCut:	<i>int32</i>	{ read-only }	1.0	open, claim, & enable
RecBarCodeRotationList:	<i>string</i>	{ read-only }	1.0	open
RecBitmapRotationList:	<i>string</i>	{ read-only }	1.7	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 }	1.0	open, claim, & enable
SlpLineCharsList:	<i>string</i>	{ read-only }	1.0	open
SlpLineHeight:	<i>int32</i>	{ read-write }	1.0	open, claim, & enable
SlpLineSpacing:	<i>int32</i>	{ read-write }	1.0	open, claim, & enable
SlpLineWidth:	<i>int32</i>	{ read-only }	1.0	open, claim, & enable
SlpLetterQuality:	<i>boolean</i>	{ read-write }	1.0	open, claim, & enable
SlpEmpty:	<i>boolean</i>	{ read-only }	1.0	open, claim, & enable
SlpNearEnd:	<i>boolean</i>	{ read-only }	1.0	open, claim, & enable
SlpSidewaysMaxLines:	<i>int32</i>	{ read-only }	1.0	open, claim, & enable
SlpSidewaysMaxChars:	<i>int32</i>	{ read-only }	1.0	open, claim, & enable
SlpMaxLines:	<i>int32</i>	{ read-only }	1.0	open, claim, & enable
SlpLinesNearEndToEnd:	<i>int32</i>	{ read-only }	1.0	open, claim, & enable
SlpBarCodeRotationList:	<i>string</i>	{ read-only }	1.1	open
SlpBitmapRotationList:	<i>string</i>	{ read-only }	1.7	open
SlpPrintSide:	<i>int32</i>	{ read-only }	1.5	open, claim, & enable

Properties (Continued)

<i>Specific (continued)</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
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**

<i>Name</i>	<i>Version</i>
open (logicalDeviceName: <i>string</i>): void { raises-exception }	1.0
close (): void { raises-exception, use after open }	1.0
claim (timeout: <i>int32</i>): void { raises-exception, use after open }	1.0
release (): void { raises-exception, use after open, claim }	1.0
checkHealth (level: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.0
clearInput (): void { }	<i>Not supported</i>
clearOutput (): void { raises-exception, use after open, claim }	1.0
directIO (command: <i>int32</i>, inout data: <i>int32</i>, inout obj: <i>object</i>): void { raises-exception, use after open }	1.0
resetStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
retrieveStatistics (inout statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
updateStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8

Specific

<i>Name</i>	<i>Version</i>
beginInsertion (timeout: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.0
beginRemoval (timeout: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.0
changePrintSide (side: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.5
cutPaper (percentage: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.0
endInsertion (): void { raises-exception, use after open, claim, enable }	1.0
endRemoval (): void { raises-exception, use after open, claim, enable }	1.0
markFeed (side: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.5

Methods (Continued)

printBarcode (station: <i>int32</i> , data: <i>string</i> , symbology: <i>int32</i> , height: <i>int32</i> , width: <i>int32</i> , alignment: <i>int32</i> , textPosition: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.0
printBitmap (station: <i>int32</i> , fileName: <i>string</i> , width: <i>int32</i> , alignment: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.0
printImmediate (station: <i>int32</i> , data: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.0
printNormal (station: <i>int32</i> , data: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.0
printTwoNormal (station: <i>int32</i> , data1: <i>string</i> , data2: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.0
rotatePrint (station: <i>int32</i> , rotation: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.0
setBitmap (bitmapNumber: <i>int32</i> , station: <i>int32</i> , fileName: <i>string</i> , width: <i>int32</i> , alignment: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.0
setLogo (location: <i>int32</i> , data: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.0
transactionPrint (station: <i>int32</i> , control: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.1
validateData (station: <i>int32</i> , data: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.1

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>
upos::events::DataEvent		<i>Not Supported</i>	
upos::events::DirectIOEvent			1.0
EventNumber:	<i>int32</i>	{ read-only }	
Data:	<i>int32</i>	{ read-write }	
Obj:	<i>object</i>	{ read-write }	
upos::events::ErrorEvent			1.0
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			1.0
OutputID:	<i>int32</i>	{ read-only }	
upos::events::StatusUpdateEvent			1.0
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

Updated in Release 1.8

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.)

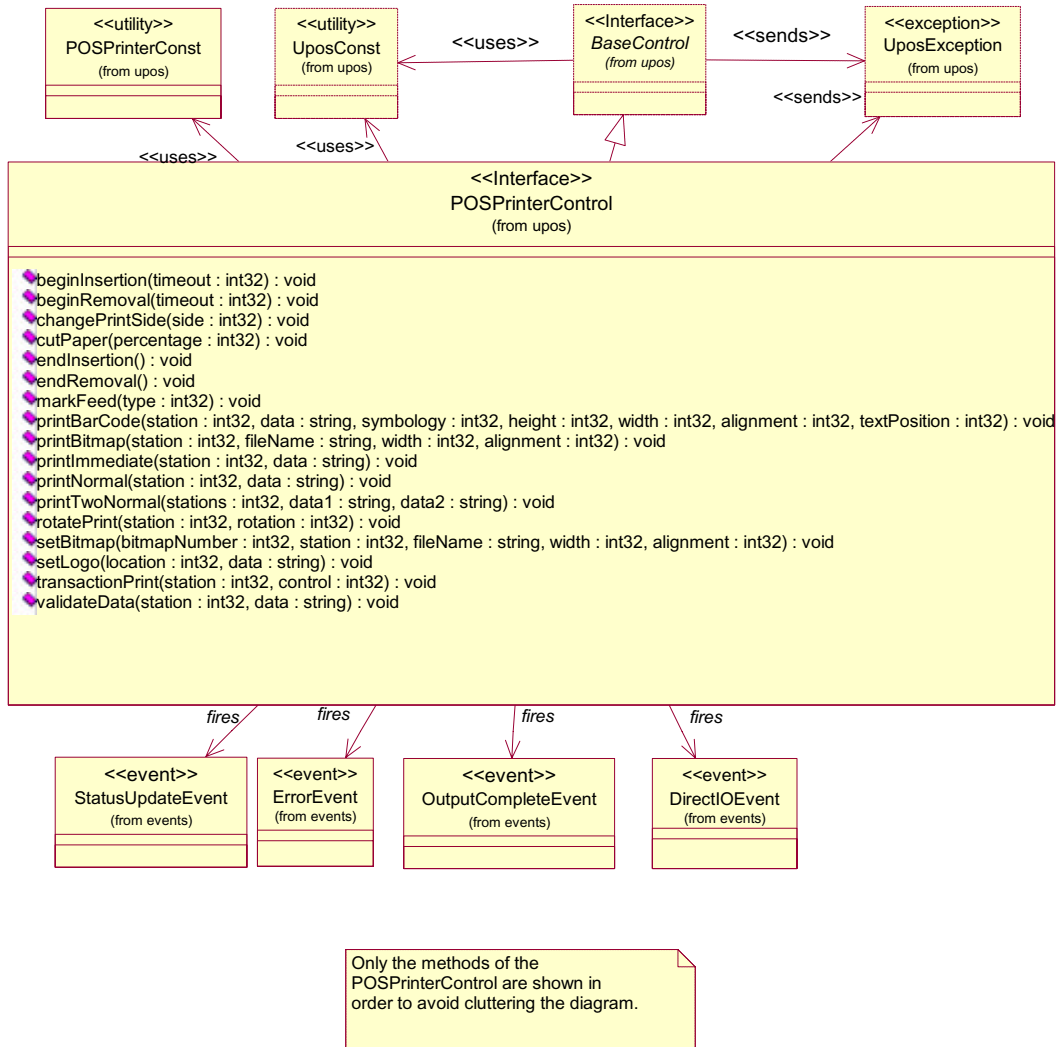
The POS printer may have several additional capabilities. See the capabilities properties for specific information.

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.
- Pixel-level printing is only supported through bitmaps when the **printBitmap** or **setBitmap** method is called with the *width* parameter set to PTR_BM_ASIS. Therefore, it is possible for the application to programmatically prepare and print bitmaps with the required pixels set.

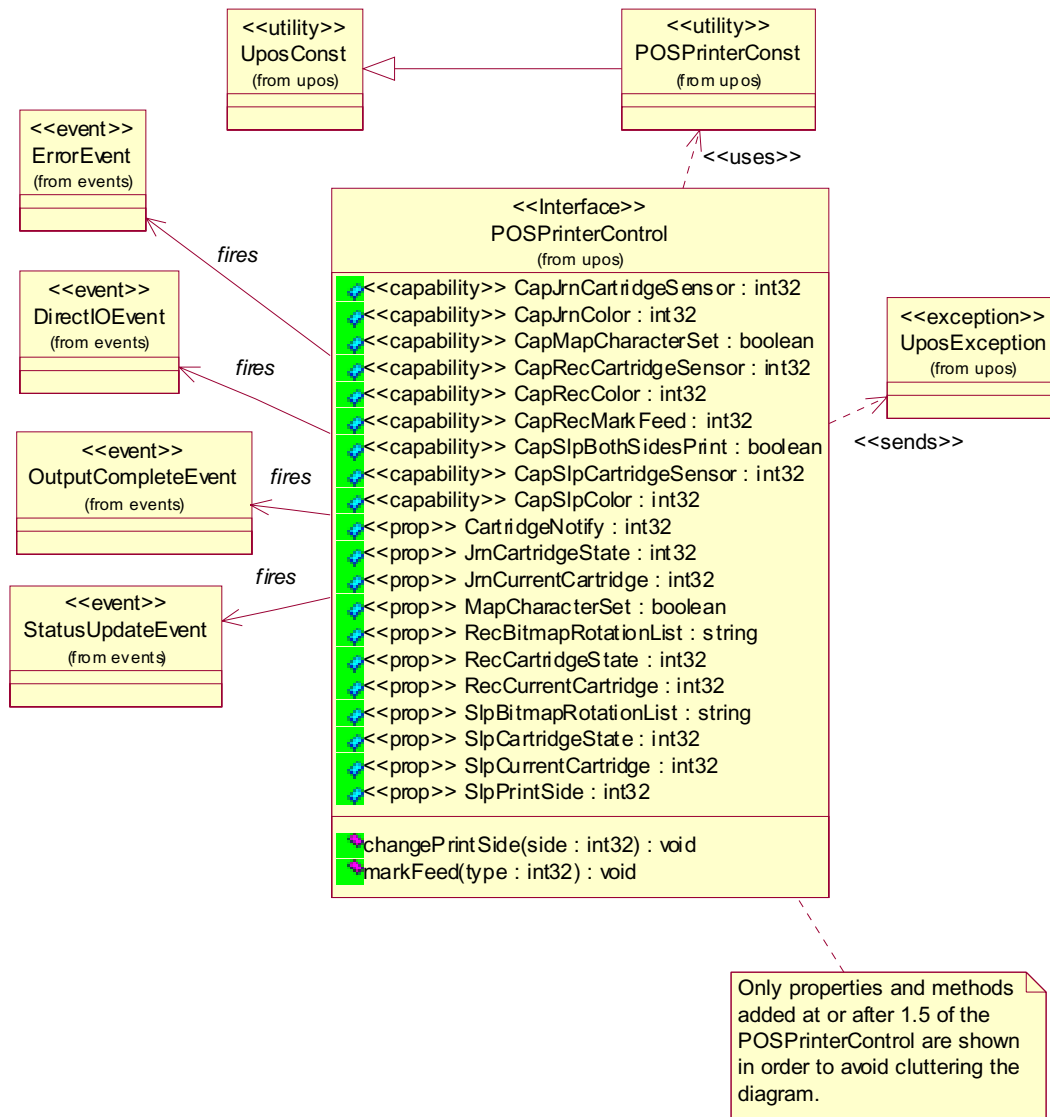
POS Printer Class Diagram

The following diagram shows the relationships between the POS Printer classes.



POS Printer Class Diagram Updates *Updated in Release 1.7*

The following diagram shows the relationships between the POS Printer classes that were updated/added in versions 1.5 and later of the specification.



Model

Updated in Release 1.8

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 in program memory, for delivery to the Physical Device as soon as the Physical Device can receive and process it, 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 buffered output data, including all asynchronous output, 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, cover open, and various cartridge handling 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.

While the printer is enabled, the printer state is monitored, and changes are reported to the application. Most printer statuses are reported by both firing a **StatusUpdateEvent** and by updating a printer property. Statuses, as defined in the later properties and events sections, are:

StatusUpdateEvent	Property
<code>PTR_SUE_COVER_OPEN</code>	CoverOpen = true
<code>PTR_SUE_COVER_OK</code>	CoverOpen = false
<code>PTR_SUE_JRN_EMPTY</code>	JrnEmpty = true
<code>PTR_SUE_JRN_NEAREMPTY</code>	JrnNearEnd = true
<code>PTR_SUE_JRN_PAPEROK</code>	JrnEmpty = JrnNearEnd = false
<code>PTR_SUE_REC_EMPTY</code>	RecEmpty = true
<code>PTR_SUE_REC_NEAREMPTY</code>	RecNearEnd = true

Release 1.8 and later

PTR_SUE_JRN_COVER_OPEN	CoverOpen = true
PTR_SUE_JRN_COVER_OK	CoverOpen = false if all covers closed; CoverOpen = true if any other cover is open
PTR_SUE_REC_COVER_OPEN	CoverOpen = true
PTR_SUE_REC_COVER_OK	CoverOpen = false if all covers closed; CoverOpen = true if any other cover is open
PTR_SUE_SLP_COVER_OPEN	CoverOpen = true
PTR_SUE_SLP_COVER_OK	CoverOpen = false if all covers closed; CoverOpen = true if any other cover is open

Release 1.8 – Clarification

The printer's slip station statuses must be reported independently from the slip insertion and removal methods – **beginInsertion** / **endInsertion** and **beginRemoval** / **endRemoval**. This is important because some applications base logic decisions upon printer state changes. That is, the application will only perform slip insertion after knowing that a slip has been placed at the entrance to the slip station. An example: After the Total key is pressed, the application enters tendering mode. It begins to monitor peripherals and the keyboard to determine the type of tender to perform. If a credit or debit card is swiped at an MSR, then its **DataEvent** causes the application to begin credit/debit tender. But if a form is placed at the slip station, then its **StatusUpdateEvent** or **SlpEmpty** property change causes the application to begin a check MICR read.

When a form is placed at the entrance to the slip station, the printer must fire a PTR_SUE_SLP_PAPEROK **StatusUpdateEvent** and set the **SlpEmpty** and **SlpNearEnd** properties to false. The application may then call the **beginInsertion** and **endInsertion** methods with reasonable confidence that they will succeed. Note that it must not be assumed that the form is ready for printing after the PTR_SUE_SLP_PAPEROK is received. Only after successful **beginInsertion** and **endInsertion** calls is the form ready for printing.

When a form is removed from the slip station, the printer must fire a PTR_SUE_SLP_EMPTY **StatusUpdateEvent** and set the **SlpEmpty** property to true. If the **beginInsertion** and **endInsertion** method sequence has not been called, then removing the form from the slip station entrance will cause this to occur. If this method sequence has successfully completed, then the event and property change will typically occur after a **beginRemoval** and **endRemoval** method sequence. But they would also occur if the slip prints beyond the end of the form or if the form is forcibly removed.

Exception: The design of some printers makes it impossible for a service to determine the presence of a form until the printer “jaws” are opened, which occurs when **beginInsertion** is called. This exception is largely limited to cases where the **CapSlpFullslip** property is false, indicating a “validation” type of slip station. Validation stations typically use the same printer mechanism as the receipt and/or journal stations. In these cases, the slip status events must be fired as soon as possible, given the constraints of the device.

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.

Release 1.8 and later – Synchronous Printing

Prior to Release 1.8 the behavior of line printers, such as thermal printers, when in synchronous mode was not clearly defined. For example, when an application called **printNormal** (PTR_S_RECEIPT, "UnifiedPOS"), the synchronous model stated that the method should not return successfully unless the text was printed on the paper. However, this example would not print on paper unless a line feed or carriage return is included in the printed data or unless the current print line was full. Starting with Release 1.8, partial line printing in synchronous mode will raise an exception. It is recommended that the application end each **printNormal**, **printTwoNormal**, or **printImmediate** call containing printable data with a newline. If partial lines are required, then transaction or asynchronous mode printing must be used.

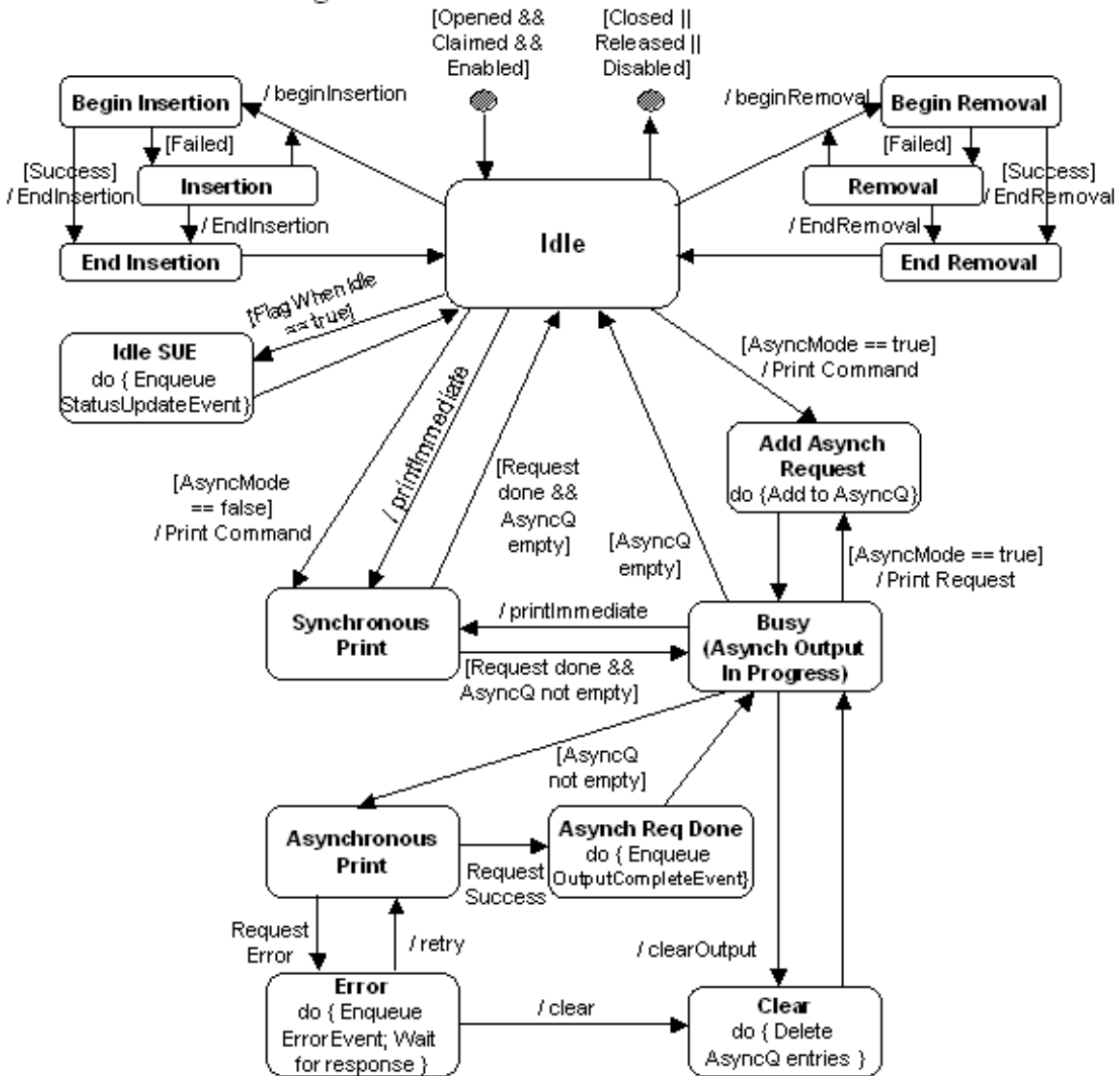
Device Sharing

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

POS Printer State Diagram



Print Commands: changePrintSide, cutPaper, markFeed, printBar Code, printBitmap, printNormal, printTwoNormal, rotatePrint

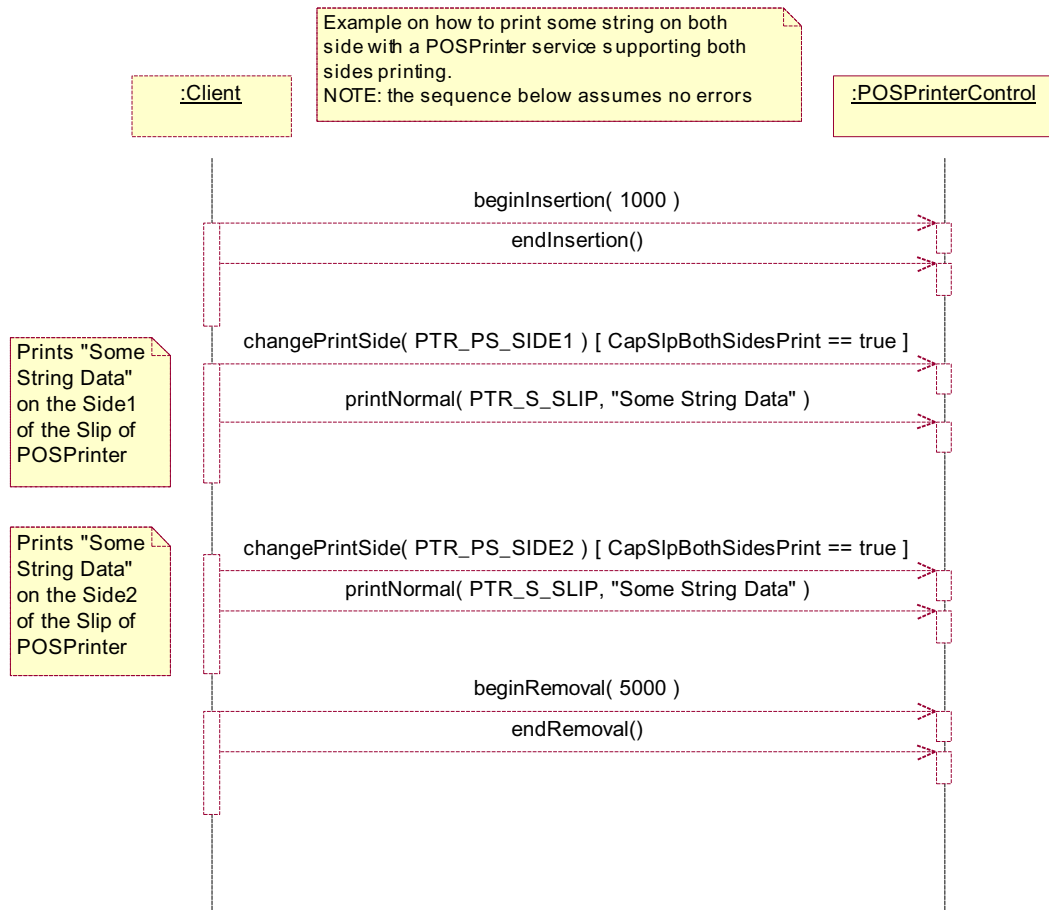
Not Shown: Validation of APIs. If an API fails during validation, then it may return an error result and return prematurely.

Special Handling:

- Sideways rotatePrint and transactions: These buffer up their data, then perform one print request.
- Status change: StatusUpdateEvents may be detected and enqueued from any state. They do not cause state transitions.

“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

Updated in Release 1.7

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.

If a sequence does not begin with ESC “|”, or it begins with ESC “|” but is not a valid UnifiedPOS escape sequence, the Service will make a reasonable effort to pass it through to the printer. However, not all such sequences can be distinguished from printable data, so unexpected results may occur.

Starting with Release 1.7, the application can use the ESC|#E escape sequence to ensure more reliable handling of the amount of data to be passed through to the printer. Use of this escape sequence will make an application non-portable. The application may, however, maintain portability by performing Embedded Data Escape sequence calls within conditional code. This code may be based upon the value of the **DeviceServiceDescription**, the **PhysicalDeviceDescription**, or the **PhysicalDeviceName** property.

NOTE: This command sequence definition and the corresponding definition in the Point Card Reader Writer Chapter, are the only known deviations from preserving the interchangeability of devices defined in this specification. If an application finds it necessary to utilize this command sequence, please inform the UnifiedPOS Committee (www.nrf-arts.org) with the details of its usage, so that a possible standard/generic Application Interface may be incorporated into a future release of the UnifiedPOS Standard. In order to preserve peripheral independence and interoperability at the Application level, it is the Committee’s position that this command sequence should be used only as a “last resort”.

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. *Updated in Release 1.8*

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 #fL	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.
Pass through embedded data (See ^a below.)	ESC #E	Send the following # characters of data through to the hardware without modifying it. The character '#' is replaced by an ASCII decimal string telling the number of bytes following the escape sequence that should be passed through as-is to the hardware.

a. This escape sequence is only available in Version 1.7 and later.

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 and wide	ESC 1C	Prints normal size.
Double wide	ESC 2C	Prints double-wide characters.
Double high	ESC 3C	Prints double-high characters.
Double high and 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 (See ^a below.)	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.
SubScript (See ^a below.)	ESC tbC	Prints SubScript characters.
SuperScript (See ^a below.)	ESC tpC	Prints SuperScript characters.
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.

a. 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 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.

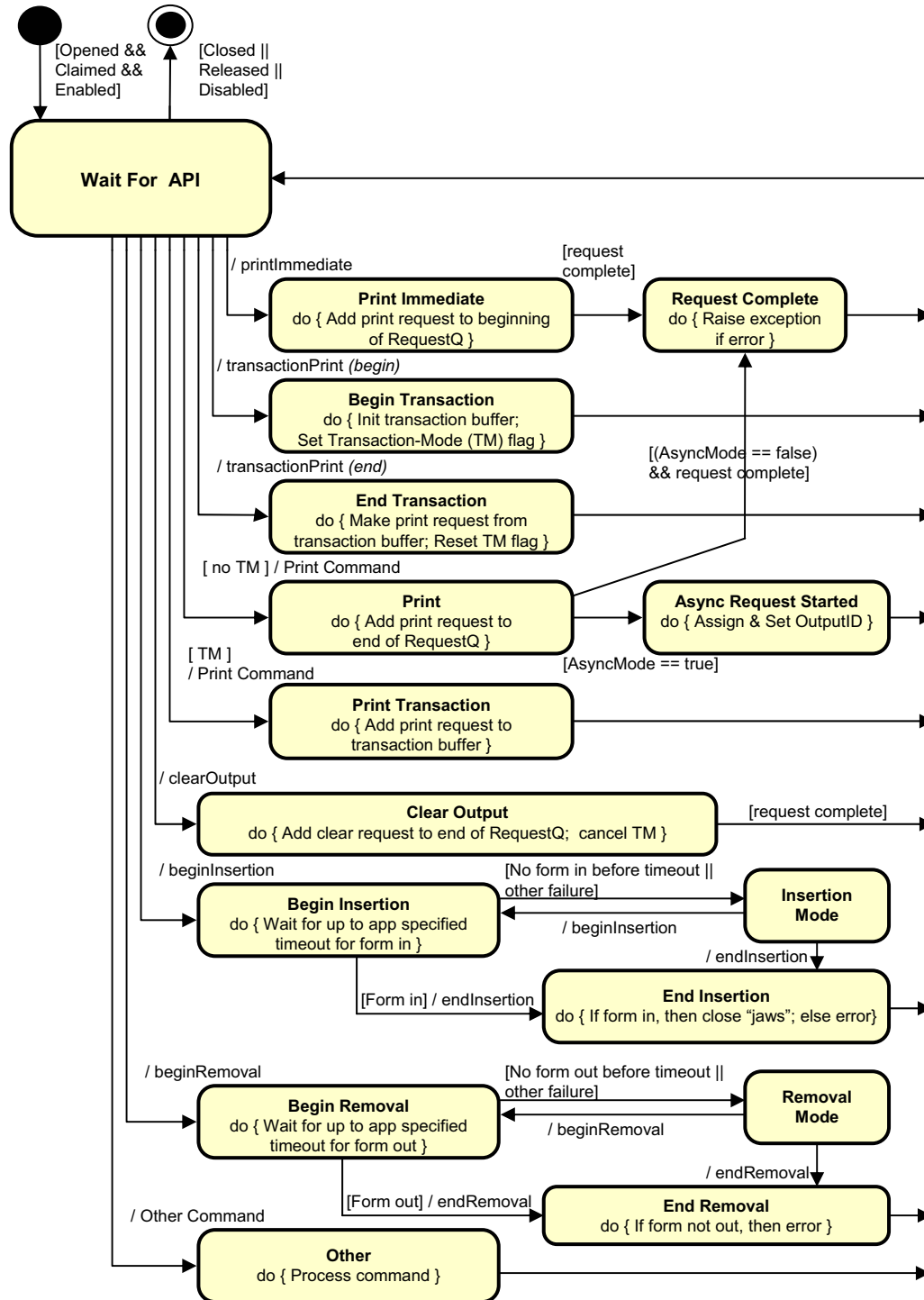
These diagrams assume:

- 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.

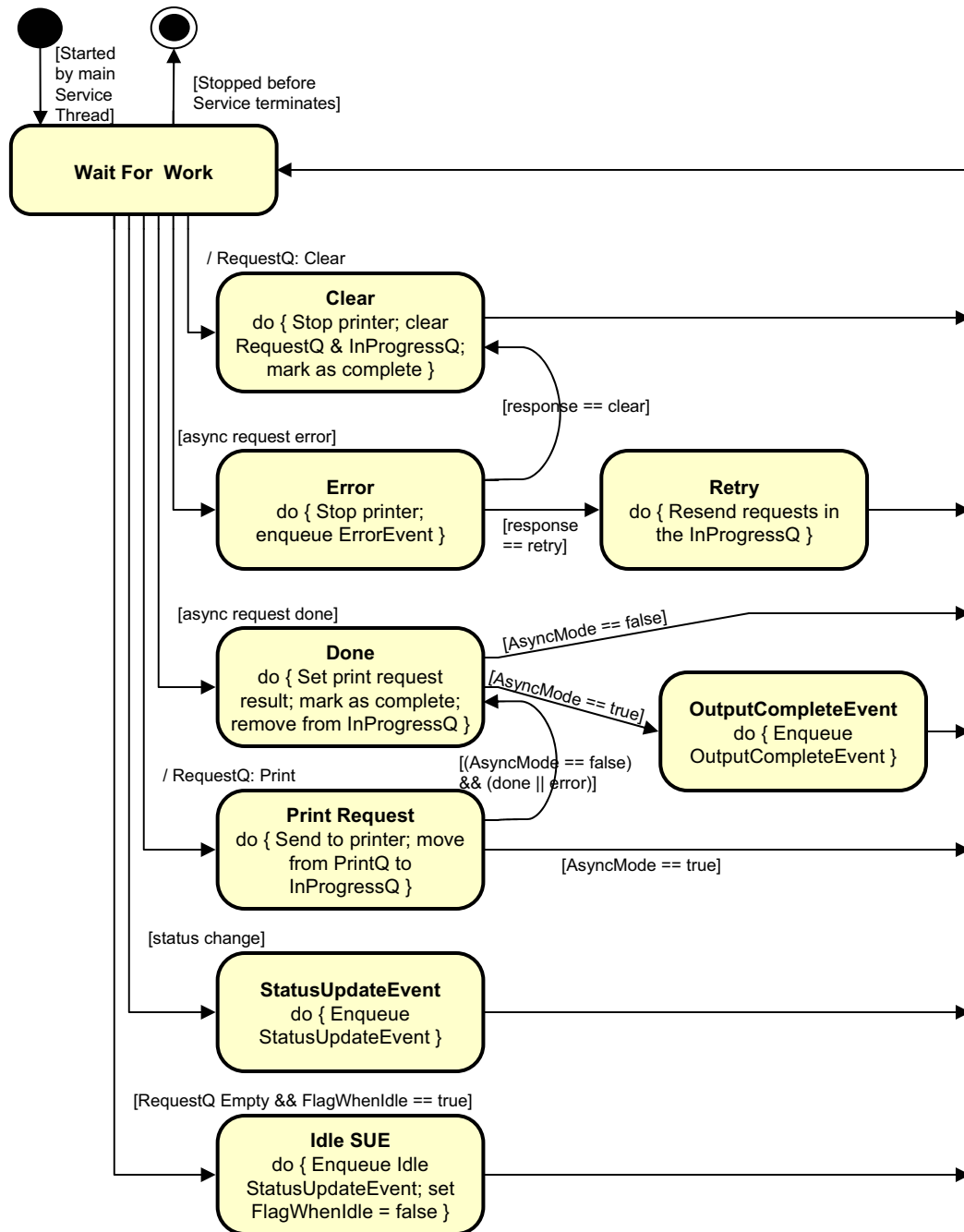
Print Commands: changePrintSide, cutPaper, markFeed, printBarCode, printBitmap, printNormal, printTwoNormal, rotatePrint.

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.

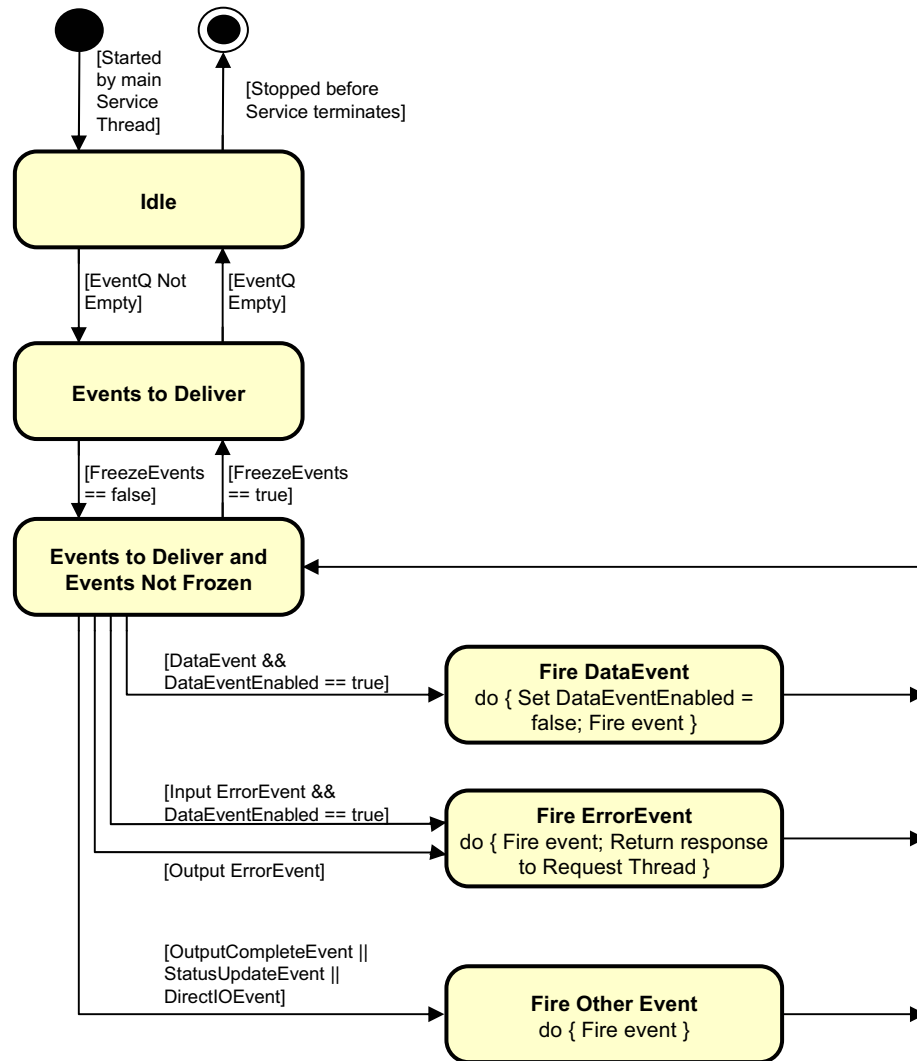
POS Printer State Diagram (Low Level): API



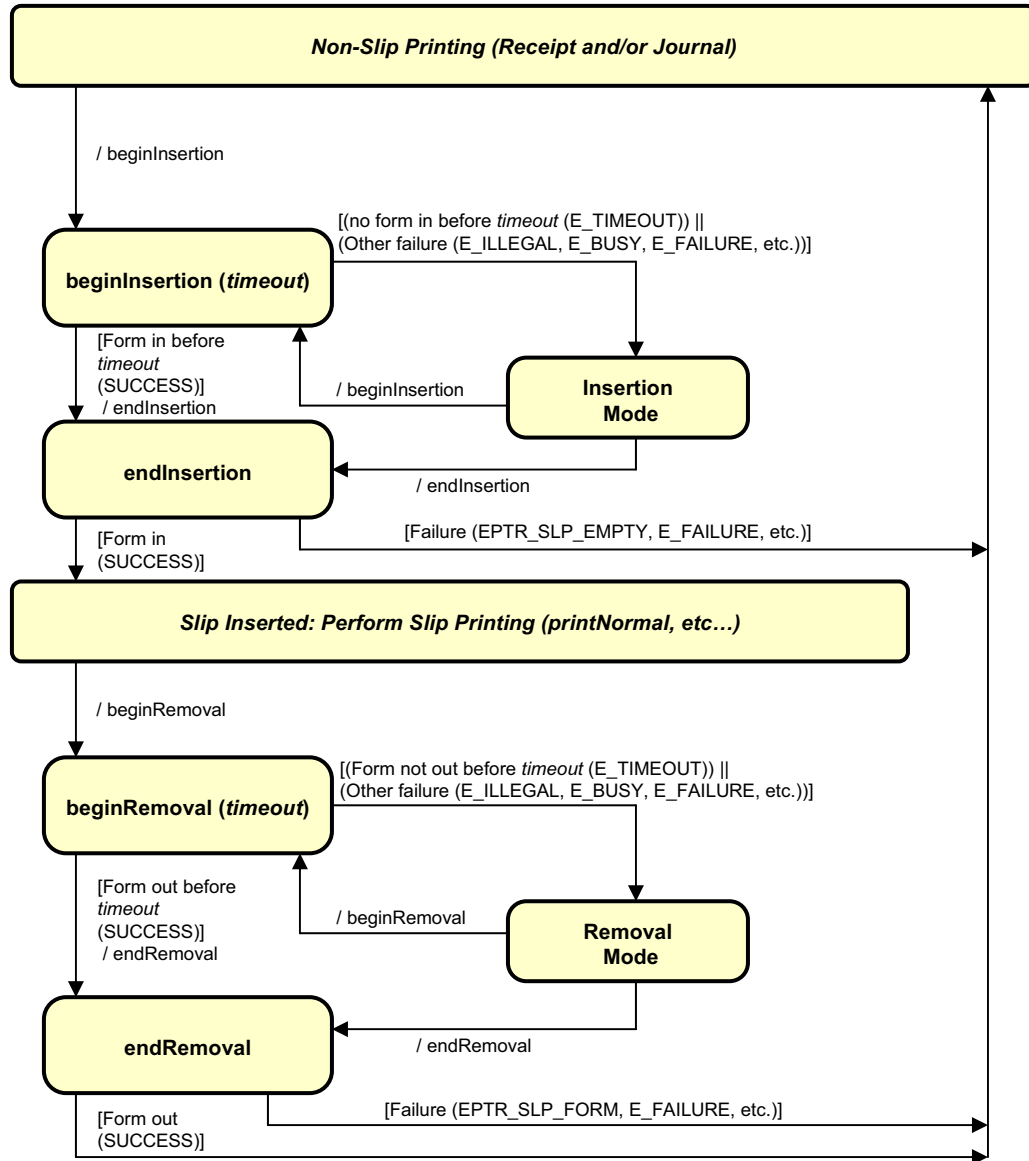
POS Printer State Diagram (Low Level): Request Thread



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	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. 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 16.

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> <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 16.												
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 16.

CapConcurrentJrnSlip Property

Syntax	CapConcurrentJrnSlip: <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 16.

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 16.

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 16.

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 16.

CapJrnBold Property

Syntax	CapJrnBold: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the journal can print bold 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 16.

CapJrnCartridgeSensor Property**Added in Release 1.5**

Syntax	CapJrnCartridgeSensor: <i>int32</i> { read-only, access after open }										
Remarks	This bit mapped parameter is used to indicate the presence of Journal Cartridge monitoring sensors. If CapJrnPresent is false, this property is “0”. Otherwise it is a logical OR combination of any of the following values: <table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>PTR_CART_REMOVED</td> <td>There is a function to indicate that the Cartridge has been removed.</td> </tr> <tr> <td>PTR_CART_EMPTY</td> <td>There is a function to indicate that the Cartridge is empty.</td> </tr> <tr> <td>PTR_CART_CLEANING</td> <td>There is a function to indicate that the head is being cleaned.</td> </tr> <tr> <td>PTR_CART_NEAREND</td> <td>There is a function to indicate that the color Cartridge is near end.</td> </tr> </tbody> </table> <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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.										
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 16.

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 16.

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 16.

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 16.

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 16.

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 16.

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 16.

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 16.

CapMapCharacterSet Property

Added in Release 1.7

Syntax	CapMapCharacterSet: <i>boolean</i> { read-only, access after open }
Remarks	<p>Defines the ability of the Service to map the characters of the application to the selected character set when printing data.</p> <p>If CapMapCharacterSet is true, then the Service is able to map the characters to the character sets defined in CharacterSetList.</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 16.
See Also	CharacterSet Property, MapCharacterSet Property, CharacterSetList Property.

CapRec2Color Property

Syntax	CapRec2Color: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, then the receipt 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 16.

CapRecBarCode Property

Syntax	CapRecBarCode: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, then the receipt has bar code printing capability.</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 16.

CapRecBitmap Property

Syntax	CapRecBitmap: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, then the receipt can print bitmaps.</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 16.

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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

CapRecCartridgeSensor Property**Added in Release 1.5**

Syntax	CapRecCartridgeSensor: <i>int32</i> { 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: <table border="1" data-bbox="467 898 1390 1260"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>PTR_CART_REMOVED</td> <td>There is a function to indicate that the Cartridge has been removed.</td> </tr> <tr> <td>PTR_CART_EMPTY</td> <td>There is a function to indicate that the Cartridge is empty.</td> </tr> <tr> <td>PTR_CART_CLEANING</td> <td>There is a function to indicate that the head is being cleaned.</td> </tr> <tr> <td>PTR_CART_NEAREND</td> <td>There is a function to indicate that the color Cartridge is near end.</td> </tr> </tbody> </table> <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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.										
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 UposException 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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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 16.

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 16.

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 16.

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 16.

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 16.

CapRecMarkFeed Property**Added in Release 1.5****Syntax** **CapRecMarkFeed: *int32* { read-only, access after open }****Remarks** This parameter indicates the type of mark sensed paper handling available.**CapRecMarkFeed** is a logical OR combination of the following values. (The values are identical to those used with the **markFeed** method.)

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.

If **CapRecMarkFeed** equals "0", mark sensed paper handling is not supported.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** **markFeed** Method.**CapRecNearEndSensor Property****Syntax** **CapRecNearEndSensor: *boolean* { read-only, access after open }****Remarks** If true, then the receipt 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 16.

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 16.

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 16.

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 16.

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 16.

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 16.

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 16.

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 16.

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 16.

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 16.

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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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> <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>	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.										
	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	SlpCartridgeState Property, SlpCurrentCartridge Property, CartridgeNotify Property.										

CapSlpColor Property**Added in Release 1.5****Syntax** **CapSlpColor: *int32* { read-only, access after open }**

Remarks This capability indicates the availability of Slip printing color cartridges.

If **CapSlpPresent** is false, this property is “0”. Otherwise, this property indicates the supported color cartridges.

CapSlpColor is a logical OR combination of any of the following values:

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.

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

Remarks If true, then the slip 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 16.

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 16.

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 16.

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 16.

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 16.

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 16.

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 16.

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 16.

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 16.

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 16.

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 16.

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 16.

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 16.

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 16.

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 16.
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 16.

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 16.								

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 16.										

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 16.

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 16.

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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	“Data Characters and Escape Sequences” on page 638.

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 border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <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> </tbody> </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 16.

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 16.

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 16.
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 16.
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 16.

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 16.

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 16.

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 16.
See Also	JrnEmpty Property.

MapCharacterSet Property**Added in Release 1.7**

Syntax	MapCharacterSet: <i>boolean</i> { read-write, access after open }
Remarks	<p>If MapCharacterSet is true and when outputting data, the Service maps the characters transferred by the application to the character set selected in the CharacterSet property for printing data.</p> <p>If MapCharacterSet is false, then no mapping is supported. In such a case the application has to ensure the mapping of the character set used in the application to the character set selected in the CharacterSet property.</p> <p>If CapMapCharacterSet is false, then this property is always 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 16.
See Also	CharacterSet Property, CapMapCharacterSet Property.

MapMode Property

Syntax	MapMode: <i>int32</i> { read-write, access after open }										
Remarks	<p>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:</p> <table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>PTR_MM_DOTS</td> <td>The printer’s dot width. This width may be different for each printer station.¹</td> </tr> <tr> <td>PTR_MM_TWIPS</td> <td>1/1440 of an inch.</td> </tr> <tr> <td>PTR_MM_ENGLISH</td> <td>0.001 inch.</td> </tr> <tr> <td>PTR_MM_METRIC</td> <td>0.01 millimeter.</td> </tr> </tbody> </table> <p>Setting this property may also change JrnLineHeight, JrnLineSpacing, JrnLineWidth, RecLineHeight, RecLineSpacing, RecLineWidth, SlpLineHeight, SlpLineSpacing, and SlpLineWidth.</p> <p>This property is initialized to PTR_MM_DOTS when the device is first enabled following the open method.</p>	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.
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.										
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.										

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**Updated in Release 1.7**

Syntax	RecBarCodeRotationList: <i>string</i> { read-only, access after open }										
Remarks	Holds the directions in which a receipt bar code 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:										
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <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> </tbody> </table>	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.										
	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 16.										
See Also	RotateSpecial Property, printBarCode Method, rotatePrint Method.										

RecBitmapRotationList Property**Added in Release 1.7**

Syntax	RecBitmapRotationList: <i>string</i> { read-only, access after open }										
Remarks	Holds the directions in which a receipt bitmap may be rotated. The string consists of rotation strings separated by commas. An empty string indicates that bitmap printing is not supported. The legal rotation strings are:										
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Bitmap may be printed in the normal orientation.</td> </tr> <tr> <td>R90</td> <td>Bitmap may be rotated 90° to the right.</td> </tr> <tr> <td>L90</td> <td>Bitmap may be rotated 90° to the left.</td> </tr> <tr> <td>180</td> <td>Bitmap may be rotated 180° - upside down.</td> </tr> </tbody> </table>	Value	Meaning	0	Bitmap may be printed in the normal orientation.	R90	Bitmap may be rotated 90° to the right.	L90	Bitmap may be rotated 90° to the left.	180	Bitmap may be rotated 180° - upside down.
Value	Meaning										
0	Bitmap may be printed in the normal orientation.										
R90	Bitmap may be rotated 90° to the right.										
L90	Bitmap may be rotated 90° to the left.										
180	Bitmap may be rotated 180° - upside down.										
	For example, if the string is “0,180”, then the printer can print normal bitmaps and upside down 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 16.										
See Also	printBitmap Method, rotatePrint 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 16.

See Also **RecNearEnd** Property.

RecLetterQuality Property

Syntax **RecLetterQuality: *boolean* { read-write, access after open-claim-enable }**

Remarks If true, prints in high quality mode. If false, prints in high speed mode.

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.
- Bitmaps may be printed in a high-density graphics mode for high-quality, and in a low-density mode for high speed.

Setting this property may also update **RecLineWidth**, **RecLineHeight**, and **RecLineSpacing** if **MapMode** is PTR_MM_DOTS. (See the footnote at **MapMode**.)

This property is initialized to false 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 16.

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 16.
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 16.
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 16.
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 16.

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 16.

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 16.

RecNearEnd Property

Syntax	RecNearEnd: <i>boolean</i> { read-only, access after open-claim-enable }
Remarks	If true, the receipt paper is low. If false, receipt paper is not low. If CapRecNearEndSensor 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 16.
See Also	RecEmpty Property.

RecSidewaysMaxChars Property

Syntax	RecSidewaysMaxChars: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	Holds the maximum number of characters that may be printed on each line in sideways mode. If CapRecLeft90 and CapRecRight90 are both false, then this property is zero. Changing the properties RecLineHeight , RecLineSpacing , and RecLineChars may cause this property to change. This property is initialized when the device is first enabled following the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	RecSidewaysMaxLines Property.

RecSidewaysMaxLines Property

Syntax	RecSidewaysMaxLines: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	Holds the maximum number of lines that may be printed in sideways mode. If CapRecLeft90 and CapRecRight90 are both false, then this property is zero. Changing the properties RecLineHeight , RecLineSpacing , and RecLineChars may cause this property to change. This property is initialized when the device is first enabled following the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
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> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>PTR_RP_NORMAL</td> <td>Print subsequent bar codes in normal orientation.</td> </tr> <tr> <td>PTR_RP_RIGHT90</td> <td>Rotate printing 90° to the right (clockwise)</td> </tr> <tr> <td>PTR_RP_LEFT90</td> <td>Rotate printing 90° to the left (counter-clockwise)</td> </tr> <tr> <td>PTR_RP_ROTATE180</td> <td>Rotate printing 180°, that is, print upside-down</td> </tr> </tbody> </table> <p>This property is initialized to PTR_RP_NORMAL by the 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 16.										
See Also	printBarcode Method.										

SlpBarcodeRotationList Property

Updated in Release 1.7

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> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <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> </tbody> </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 16.										
See Also	RotateSpecial Property, printBarcode Method, rotatePrint Method.										

SlpBitmapRotationList Property***Added in Release 1.7*****Syntax** **SlpBitmapRotationList:** *string* { read-only, access after open }**Remarks** Holds the directions in which a slip bitmap may be rotated. The string consists of rotation strings separated by commas. An empty string indicates that bitmap printing is not supported. The legal rotation strings are:

Value	Meaning
0	Bitmap may be printed in the normal orientation.
R90	Bitmap may be rotated 90° to the right.
L90	Bitmap may be rotated 90° to the left.
180	Bitmap may be rotated 180° - upside down.

For example, if the string is “0,180”, then the printer can print normal bitmaps and upside down 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 16.**See Also** **printBitmap** Method, **rotatePrint** 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 Slp 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 UposException 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 16.**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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
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 16.
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 16.
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 16.
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 UposException may be thrown when this property is accessed. For further information, see "Errors" on page 16.

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 UposException may be thrown when this property is accessed. For further information, see "Errors" on page 16.

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 UposException may be thrown when this property is accessed. For further information, see "Errors" on page 16.

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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	SlpEmpty Property, SlpLinesNearEndToEnd Property.

SlpPrintSide Property**Added in Release 1.5**

Syntax	SlpPrintSide: int32 { 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 inserted, 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 border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <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> </tbody> </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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	SlpSidewaysMaxChars Property.

E_EXTENDED *ErrorCodeExtended* = 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 16.

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	<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 is out of paper. (Can only apply if AsyncMode is false.)</p>

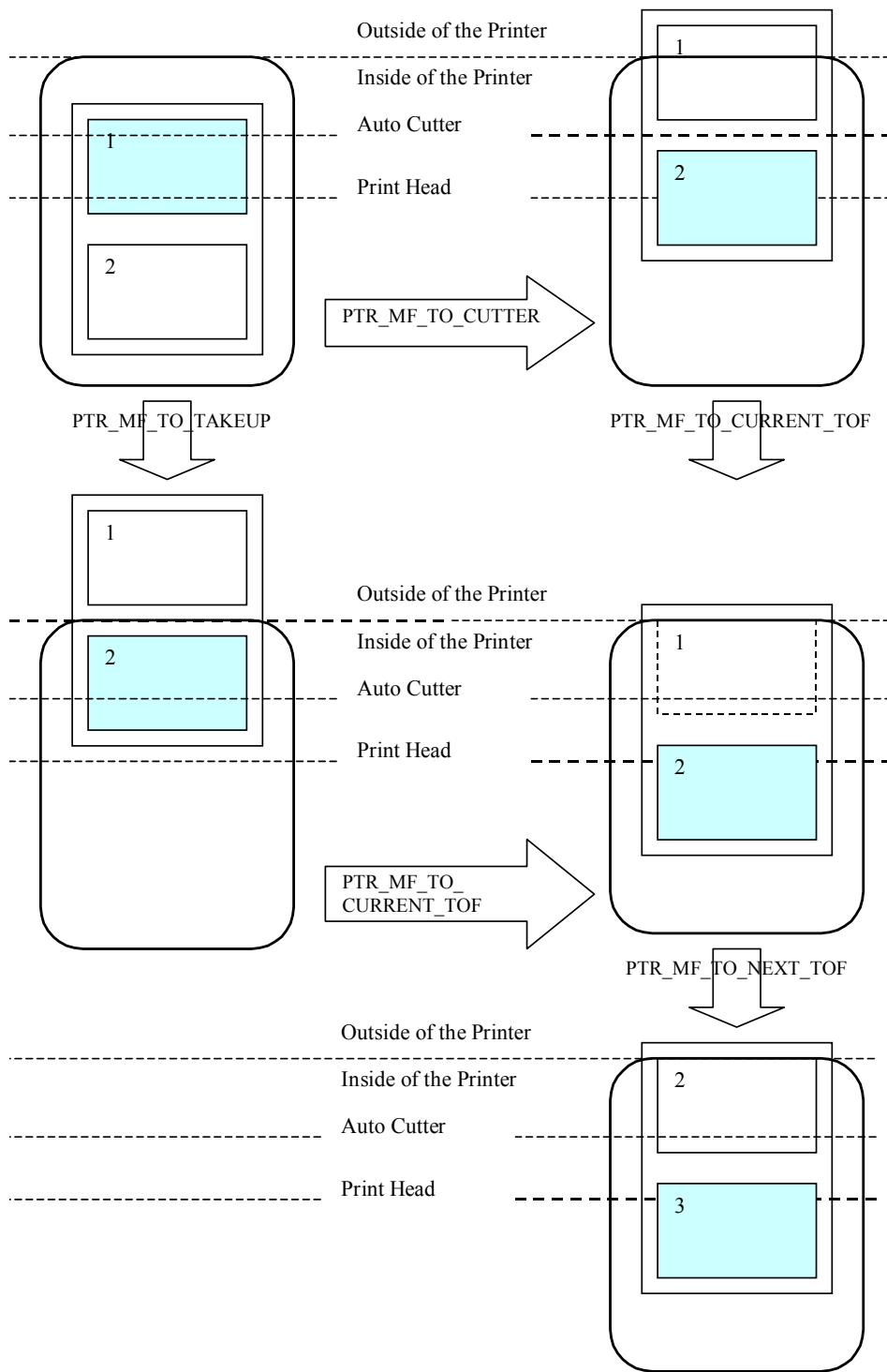
See Also “Data Characters and Escape Sequences” on page 638.

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 16.</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_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> <p><i>ErrorCodeExtended</i> = EPTR_COVER_OPEN: The device was taken out of insertion mode while the printer cover was open.</p> <p><i>ErrorCodeExtended</i> = EPTR_SLP_EMPTY: The device was taken out of insertion mode without a form being inserted.</p> </td> </tr> </tbody> </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	<p><i>ErrorCodeExtended</i> = EPTR_COVER_OPEN: The device was taken out of insertion mode while the printer cover was open.</p> <p><i>ErrorCodeExtended</i> = EPTR_SLP_EMPTY: The device was taken out of insertion mode without a form being inserted.</p>
Value	Meaning								
E_BUSY	Cannot perform request while output is in progress.								
E_ILLEGAL	The printer is not in slip insertion mode.								
E_EXTENDED	<p><i>ErrorCodeExtended</i> = EPTR_COVER_OPEN: The device was taken out of insertion mode while the printer cover was open.</p> <p><i>ErrorCodeExtended</i> = EPTR_SLP_EMPTY: The device was taken out of insertion mode without a form being inserted.</p>								
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 16.</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_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> </tbody> </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.								



Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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	The receipt print station does not support the given mark sensed paper handling function. (Refer to the CapRecMarkFeed 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 paper is empty. (Can only apply if AsyncMode is false.)

See Also **CapRecMarkFeed** Property.

printBarcode Method**Updated in Release 1.8**

Syntax **printBarcode** (*station*: *int32*, *data*: *string*, *symbology*: *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> ²	Character string to be bar coded.
<i> symbology</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.

². In the OPOS environment, the format of *data* depends upon the value of the **BinaryConversion** property. See **BinaryConversion** property on page A-28.

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

Value	Meaning
<i>One Dimensional Symbologies</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"
<u>Added in Release 1.8</u>	
PTR_BCS_Code128_Parsed	Code 128 with parsing.
PTR_BCS_RSS14	Reduced Space Symbology
PTR_BCS_RSS_EXPANDED	Reduced Space Symbology - Expanded
<i>Two Dimensional Symbologies</i>	
PTR_BCS_PDF417	PDF 417
PTR_BCS_MAXICODE	MAXICODE
<i>Special Cases</i>	
PTR_BCS_OTHER	If a Service defines additional symbologies, they will be greater or equal to this value.

Special Considerations for Code 128

The Code 128 Bar Code Symbology is comprised of three code sets and also includes some special characters that denote either a change in code set, a function code, or a shift code. The characters for each code set are:

Code Set	Character Set
Code A	0x00-0x5f, FNC1, FNC2, FNC3, FNC4, SHIFT, CODE B, CODE C
Code B	0x20-0x7f, FNC1, FNC2, FNC3, FNC4, SHIFT, CODE A, CODE C
Code C	0x00-0x63 for decimal values 00-99, FNC1, CODE A, CODE B

Release 1.7 and earlier

The data format to be supplied by the application was not specified in these releases. Therefore, the default code set and data content varies by vendor. An application that sends Code 128 data to a 1.7 or earlier service will need to conform to that service's requirements.

Release 1.8 and later

For migration of current applications, the symbology PTR_BCS_Code128 is maintained so that a service may continue to support the data format that it used with earlier releases. (New service implementations should handle this symbology as with PTR_BCS_Code128_Parsed.)

The new symbology PTR_BCS_Code128_Parsed standardizes the data format with consistent parsing. Data is comprised of ASCII characters, which the service maps to the corresponding value for the selected code set. In Code Sets A and B, this will be a one to one mapping. In Code Set C, each pair of digits is converted to a single Code C data character in the range 0x00 through 0x63 (99). (If the Code Set C data contains an odd number of digits, then a leading zero digit is added by the service before conversion.) A sentinel character, the left curly bracket "{", followed by a certain value, is used to indicate a special character. The following table lists the character pairs for encoding the special characters:

<u>Special Characters</u>	<u>ASCII Representation</u>
SHIFT	{S
CODE A	{A
CODE B	{B
CODE C	{C
FNC1	{1
FNC2	{2
FNC3	{3
FNC4	{4
{	{{

The default Code Set may differ by vendor, so a starting code set is required at the start of the data.

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 UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	One of the following parameter errors occurred: <ul style="list-style-type: none"> * <i>station</i> does not exist * <i>station</i> does not support bar code printing * <i>height</i> or <i>width</i> is zero or too big * <i>symbology</i> is not supported * not all characters in <i>data</i> are supported by <i>symbology</i> * <i>alignment</i> is invalid or too big * Code Set is not specified for PTR_BCS_Code128_Parsed at start of <i>data</i> * <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	<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>

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.)

See Also **MapMode** Property, **RotateSpecial** Property.

printBitmap Method

Updated in Release 1.7

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, gif, or jpeg files. ³
<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 <i>width</i> 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 .

³. In the **OPOS** environment, the Service Object must support two-color (black and white) uncompressed Windows bitmaps. Black pixels are printed, while white pixels are not printed. Additional formats may be supported.

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. If a partial text line has been sent (for example, via **printNormal**) but not yet printed, then an implicit newline is added to this text and the line is printed before the bitmap is printed. Text data sent after this **printBitmap** begins on the line following the bitmap.

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 `UposException` may be thrown when this method is invoked. For further information, see "Errors" on page 16.

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: <ul style="list-style-type: none"> * <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

ErrorCodeExtended = EPTR_TOOBIG:

The bitmap is either too wide to print without transformation, or it is too big to transform.

ErrorCodeExtended = EPTR_COVER_OPEN:

The printer cover is open.

(Can only apply if **AsyncMode** is false.)

ErrorCodeExtended = EPTR_BADFORMAT:

The specified file is either not a bitmap file, or it is in an unsupported format.

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.)

See Also **MapMode** Property.

Errors A `UposException` may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Updated in Release 1.7

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>
-----------------	---

<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> ⁶	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 638. The characters must all fit on one printed line, so that the printer may attempt to print on both stations simultaneously.
---------------------------	--

<i>data2</i> ⁶	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.
---------------------------	--

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.

⁶. In the OPOS environment, the format of *data1* and *data2* depends upon the value of the **BinaryConversion** property. See **BinaryConversion** property on page A-28.

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 16.

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>

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.)

See Also **printNormal** Method

limitations that may be specified by the **RecBarcodeRotationList**, **SlpBarcodeRotationList**, **RecBitmapRotationList**, and **SlpBitmapRotationList** properties respectively.

If *rotation* includes PTR_RP_BARCODE, then the contents of **RotateSpecial** are ignored.

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 16.

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 638, **RotateSpecial** Property.

setBitmap Method**Updated in Release 1.7**

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. Valid bitmap numbers are 1 through 20. Release 1.6 and earlier: Valid bitmap numbers are 1 and 2.
<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. ⁷ 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.

If a partial text line has been sent before the print bitmap escape sequence is encountered, then an implicit newline is added to this text and the line is printed before the bitmap is printed. Text data sent after the print bitmap escape sequence begins on the line following the bitmap.

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 most frequently used bitmaps should be assigned a small *bitmapNumber* (close to 1), while occasionally used bitmaps should be assigned the larger *bitmapNumbers*. The Service will use these subsets to determine how best to store the bitmaps. It may download them to the device when possible, or cache them in Service memory, or simply remember the *fileName* and associated properties for use when it is printed.

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 through 20 for each of the two valid

⁷. In the **OPOS** environment, the Service Object must support two-color (black and white) uncompressed Windows bitmaps. Black pixels are printed, while white pixels are not printed. Additional formats may be supported.

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 16.

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 638, **printBitmap** Method.

setLogo Method

Updated in Release 1.7

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> ⁸	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 16.

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 638.

⁸. In the OPOS environment, the format of this data depends upon the value of the **BinaryConversion** property. See **BinaryConversion** property on page A-28.

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.)

See Also **CapTransaction** Property, **cutPaper** Method, **printBarcode** Method, **printBitmap** Method, **printNormal** Method, **rotatePrint** Method.

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 and wide	Not supported.
Double wide	Not supported.
Double high	Not supported.
Double high and 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 638.

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 15, **directIO** Method.

ErrorEvent

Updated in Release 1.7

```
<< 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 attributes:

Attributes	Type	Description
<i>ErrorCode</i>	<i>int32</i>	Error code causing the error event. See a list of Error Codes on page 16.
<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 all buffered output data, including all asynchronous output. (The effect is the same as when clearOutput is called.) 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 21, "Device Information Reporting Model" on page 26

OutputCompleteEvent

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

Description	Notifies the application that the queued output request associated with the <i>OutputID</i> attribute has completed successfully.						
Attributes	This event contains the following attribute:						
	<table border="1"> <thead> <tr> <th>Attributes</th> <th>Type</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><i>OutputID</i></td> <td><i>int32</i></td> <td>The ID number of the asynchronous output request that is complete.</td> </tr> </tbody> </table>	Attributes	Type	Description	<i>OutputID</i>	<i>int32</i>	The ID number of the asynchronous output request that is complete.
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 21.						

StatusUpdateEvent**Updated in Release 1.8**

<< 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 is inserted, and no slip form has been detected at the entrance to the slip station. (See “Model” on page 630 for further details on slip properties and events.)	
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 63.

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.

Release 1.8 and later – Specific Cover State Reporting

Starting with Release 1.8, **StatusUpdateEvents** for specific stations' covers are supported. If a printer has only one cover or if the printer cannot determine/report which covers are open, then only the original PTR_SUE_COVER_OPEN and PTR_SUE_COVER_OK events should be fired.

For printers supporting multiple covers, the original events should also be fired for compatibility with current applications. In these cases, the station-specific event should be fired **first**, followed by the original event.

If more than one cover is open, the original PTR_SUE_COVER_OPEN event should only be fired once after a cover is opened. A PTR_SUE_COVER_OK

event should only be fired after all the covers are closed.

The event’s *Status* attribute can contain one of the following additional values to indicate a status change.

Value	Meaning
PTR_SUE_JRN_COVER_OPEN	Journal station cover is open.
PTR_SUE_JRN_COVER_OK	Journal station cover is closed.
PTR_SUE_REC_COVER_OPEN	Receipt station cover is open.
PTR_SUE_REC_COVER_OK	Receipt station cover is closed.
PTR_SUE_SLP_COVER_OPEN	Slip station cover is open.
PTR_SUE_SLP_COVER_OK	Slip station cover is closed.

Example A: Suppose that a printer includes two cover sensors, but reports “cover open” if either is open. Then here are the actions and **StatusUpdateEvents** that should be fired.

Action	StatusUpdateEvent
Open front cover	PTR_SUE_COVER_OPEN
Open rear cover	(no additional SUE)
Close front cover	(no additional SUE)
Close rear cover	PTR_SUE_COVER_OK

Example B: Suppose that a printer includes two sensors which report their statuses independently. Then here are the actions and **StatusUpdateEvents** that should be fired.

Action	StatusUpdateEvent(s)
Open front cover	PTR_SUE_SLP_COVER_OPEN, then PTR_SUE_COVER_OPEN
Open rear cover	PTR_SUE_REC_COVER_OPEN
Close front cover	PTR_SUE_SLP_COVER_OK
Close rear cover	PTR_SUE_REC_COVER_OK, then PTR_SUE_COVER_OK

This status reporting allows the migration of applications written to earlier releases, plus additional functionality for applications written to the new release:

- An application that either ignores the new statuses or was written before 1.8 continues to respond to the PTR_SUE_COVER_OPEN and PTR_SUE_COVER_OK **StatusUpdateEvents**. (It is assumed that the application will ignore statuses that are not expected.)
- An application written to support the new statuses can respond to the station-specific status (PTR_SUE_XXX_COVER_OK), and the general status (PTR_SUE_COVER_OK) will not provide any additional information. But if it receives a general status without a preceding station-specific status, then it processes the general status.

Remarks Enqueued when a significant status event has occurred.

See Also “Events” on page 15.

Remote Order Display

This Chapter defines the Remote Order Display device category.

Summary

Properties (UML attributes)

<i>Common</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</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
CapStatisticsReporting:	<i>boolean</i>	{ read-only }	1.8	open
CapUpdateStatistics:	<i>boolean</i>	{ read-only }	1.8	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

Properties (Continued)

<i>Specific</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
CapMapCharacterSet:	<i>boolean</i>	{ read-only }	1.7	open
CapSelectCharacterSet:	<i>boolean</i>	{ read-only }	1.3	open, claim, & enable
CapTone:	<i>boolean</i>	{ read-only }	1.3	open, claim, & enable
CapTouch:	<i>boolean</i>	{ read-only }	1.3	open, claim, & enable
CapTransaction:	<i>boolean</i>	{ read-only }	1.3	open
AsyncMode:	<i>boolean</i>	{ read-write }	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
CharacterSet:	<i>int32</i>	{ read-only }	1.3	open, claim, & enable
CharacterSetList:	<i>string</i>	{ read-only }	1.3	open, claim, & enable
Clocks:	<i>int32</i>	{ read-only }	1.3	open, claim, & enable
CurrentUnitID:	<i>int32</i>	{ read-write }	1.3	open, claim, & enable
ErrorString:	<i>string</i>	{ read-only }	1.3	open
ErrorUnits:	<i>int32</i>	{ read-only }	1.3	open
EventString:	<i>string</i>	{ read-only }	1.3	open & claim
EventType:	<i>int32</i>	{ read-write }	1.3	open
EventUnitID:	<i>int32</i>	{ read-only }	1.3	open & claim
EventUnits:	<i>int32</i>	{ read-only }	1.3	open & claim
MapCharacterSet:	<i>boolean</i>	{ read-write }	1.7	open
SystemClocks:	<i>int32</i>	{ read-only }	1.3	open, claim, & enable
SystemVideoSaveBuffers:	<i>int32</i>	{ read-only }	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
VideoDataCount:	<i>int32</i>	{ read-only }	1.3	open, claim, & enable
VideoMode:	<i>int32</i>	{ read-write }	1.3	open, claim, & enable
VideoModesList:	<i>string</i>	{ read-only }	1.3	open, claim, & enable
VideoSaveBuffers:	<i>int32</i>	{ read-only }	1.3	open, claim, & enable

Methods (UML operations)**Common**

<i>Name</i>	<i>Version</i>
open (logicalDeviceName: <i>string</i>): void { raises-exception }	1.3
close (): void { raises-exception, use after open }	1.3
claim (timeout: <i>int32</i>): void { raises-exception, use after open }	1.3
release (): void { raises-exception, use after open, claim }	1.3
checkHealth (level: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3
clearInput (): void { raises-exception, use after open, claim }	1.3
clearOutput (): void { raises-exception, use after open, claim }	1.3
directIO (command: <i>int32</i>, inout data: <i>int32</i>, inout obj: <i>object</i>): void { raises-exception, use after open }	1.3
resetStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
retrieveStatistics (inout statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
updateStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8

Specific

<i>Name</i>	
clearVideo (units: <i>int32</i>, attribute: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3
clearVideoRegion (units: <i>int32</i>, row: <i>int32</i>, column: <i>int32</i>, height: <i>int32</i>, width: <i>int32</i>, attribute: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3
controlClock (units: <i>int32</i>, function: <i>int32</i>, clockId: <i>int32</i>, hour: <i>int32</i>, min: <i>int32</i>, sec: <i>int32</i>, row: <i>int32</i>, column: <i>int32</i>, attribute: <i>int32</i>, mode: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3
controlCursor (units: <i>int32</i>, function: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3
copyVideoRegion (units: <i>int32</i>, row: <i>int32</i>, column: <i>int32</i>, height: <i>int32</i>, width: <i>int32</i>, targetRow: <i>int32</i>, targetColumn: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3
displayData (units: <i>int32</i>, row: <i>int32</i>, column: <i>int32</i>, attribute: <i>int32</i>, data: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.3
drawBox (units: <i>int32</i>, row: <i>int32</i>, column: <i>int32</i>, height: <i>int32</i>, width: <i>int32</i>, attribute: <i>int32</i>, bordertype: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3

Methods (Continued)

freeVideoRegion (units: <i>int32</i>, bufferId: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3
resetVideo (units: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3
restoreVideoRegion (units: <i>int32</i>, targetRow: <i>int32</i>, targetColumn: <i>int32</i>, bufferId: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3
saveVideoRegion (units: <i>int32</i>, row: <i>int32</i>, column: <i>int32</i>, height: <i>int32</i>, width: <i>int32</i>, bufferId: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3
selectCharacterSet (units: <i>int32</i>, characterSet: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3
setCursor (units: <i>int32</i>, row: <i>int32</i>, column: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3
transactionDisplay (units: <i>int32</i>, function: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3
updateVideoRegionAttribute (units: <i>int32</i>, function: <i>int32</i>, row: <i>int32</i>, column: <i>int32</i>, height: <i>int32</i>, width: <i>int32</i>, attribute: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3
videoSound (units: <i>int32</i>, frequency: <i>int32</i>, duration: <i>int32</i>, numberOfCycles: <i>int32</i>, interSoundWait: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>
upos::events::DataEvent			1.3
Status:	<i>int32</i>	{ read-only }	
upos::events::DirectIOEvent			1.3
EventNumber:	<i>int32</i>	{ read-only }	
Data:	<i>int32</i>	{ read-write }	
Obj:	<i>object</i>	{ read-write }	
upos::events::ErrorEvent			1.3
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			1.3
OutputID:	<i>int32</i>	{ read-only }	
upos::events::StatusUpdateEvent			1.3
Status:	<i>int32</i>	{ read-only }	

General Information

The Remote Order Display programmatic name is “RemoteOrderDisplay”.

Capabilities

The Remote Order Display has the following minimal set of capabilities:

- Supports color or monochrome text character displays.
- Supports 8 foreground colors (or gray scale on monochrome display) with the option of using the intensity attribute.
- Supports 8 background colors (or gray scale on monochrome display) with the option of using only a blinking attribute.
- The individual event types support disabling such that the application only receives a subset of data events if requested.
- Supports video region buffering.
- Supports cursor functions.
- Supports clock functions.
- Supports resetting a video unit to power on state.

The Remote Order Display may also have the following additional capabilities:

- Supports multiple video displays each with possibly different video modes.
- Supports touch video input for a touch screen display unit.
- Supports video enunciator output with frequency and duration.
- Supports tactile feedback via an automatic tone when a video display unit is touched (for touch screen only).
- Supports downloading alternate character sets to one or many video units.
- Supports transaction mode display output to one or many video units.

The following capability is not supported:

- Support for graphical displays, where the video display is addressable by individual pixels or dots. The addition of this support is under investigation for future revisions.

Model

Updated in Release 1.7

The general model of a Remote Order Display:

The Remote Order Display device class is a subsystem of video units. The initial targeted environment is food service, to display order preparation and fulfillment information. Remote Order Displays are often used in conjunction with Bump Bars.

The general model of a Remote Order Display is an output device but may also be an input device when, in some implementations, the device can report additional status or user input data back to the application program.

- The subsystem can support up to 32 video units.

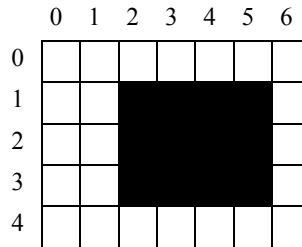
Typically, one application on one workstation (or POS Terminal) would manage and control the entire subsystem of Remote Order Displays. However, if applications on the same or other workstations (or POS Terminals) would need to access the subsystem, then one of the applications must act as a subsystem server and expose the necessary interfaces to other applications.

- All specific methods are broadcast methods. This means that the method can apply to one unit, a selection of units or all online units. The *units* parameter is an **int32**, with each bit identifying an individual video unit. The Service will attempt to satisfy the method for all units indicated in the *units* parameter. If an error is received from one or more units, the **ErrorUnits** property is updated with the appropriate units in error. The **ErrorString** property is updated with a description of the error or errors received. The method will then raise a *UposException*. In the case where two or more units encounter different errors, the exception's *ErrorCode* will indicate the more severe error.
- The common methods **checkHealth**, **clearInput**, and **clearOutput** are not broadcast methods and use the unit ID indicated in the **CurrentUnitID** property. See the description of these common methods to understand how the **CurrentUnitID** property is used.
- When the **CurrentUnitID** 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 represent a single video unit. The definitions range from **ROD_UID_1** to **ROD_UID_32**. These definitions are also used to create the bitwise parameter, *units*, used in the broadcast methods.

- The rows and columns are numbered beginning with (0,0) at the top-left corner of the video display. The dimensions are defined by the *height* and *width* parameters. The region depicted below would have the parameters *row = 1, column = 2, height = 3, and width = 4*.



All position parameters are expressed in text characters.

- The VGA-like *attribute* parameter, that is used in various methods, is an **int32**. Bits 7-0 define the text attribute and bits 31-8 are reserved and must be 0, otherwise an E_ILLEGAL exception is raised. The following table defines bits 7-0:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Blinking	Background and Color			Intensity	Foreground Color		

If a foreground or background color is requested, but the Service does not support that color, it chooses the best fit from the colors supported.

The following constants may be used, with up to one constant selected from each category:

- Blinking: ROD_ATTR_BLINK
- Background Color: ROD_ATTR_BG_color, where *color* is replaced by BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, or GRAY
- Intensity: ROD_ATTR_INTENSITY
- Foreground Color: ROD_ATTR_FG_color, where *color* is replaced by BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, or GRAY

For touch video input, the Remote Order Display Control follows the general “Input Model” for event-driven input with some differences:

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

For video and tone output, the Remote Order Display follows the general Output Model, with some enhancements:

- The following methods are always performed synchronously: **controlClock**, **controlCursor**, **selectCharacterSet**, **resetVideo**, and **setCursor**. These methods will fail if asynchronous output is outstanding. The following method is also always performed synchronously but without regard to outstanding asynchronous output: **freeVideoRegion**.
- The following methods are performed either synchronously or asynchronously, depending on the value of the **AsyncMode** property: **clearVideo**, **clearVideoRegion**, **copyVideoRegion**, **displayData**, **drawBox**, **restoreVideoRegion**, **saveVideoRegion**, **transactionDisplay**, **updateVideoRegionAttribute**, and **videoSound**. When **AsyncMode** is false, then these methods operate synchronously.

When **AsyncMode** is true, then these methods operate as follows:

- The request is buffered in program memory for delivery to the Physical Device as soon as the Physical Device can receive and process it, the **OutputID** property is set to an identifier for this request, and returns as soon as possible. When the device completes the request successfully, then the **EventUnits** property is updated and an **OutputCompleteEvent** is enqueued. A property of this event contains the output ID of the completed request.
Asynchronous methods will **not** raise a **UposException** due to a display problem, such as communications failure. These errors will only be reported by an **ErrorEvent**. A **UposException** is raised only if the display is not claimed and enabled, a parameter is invalid, or the request cannot be enqueued. The first two error cases are due to an application error, while the last is a serious system resource exception.

- If an error occurs while performing an asynchronous request, an **ErrorEvent** is enqueued. The **EventUnits** property is set to the unit or units in error. The **EventString** property is also set.

*Note: **ErrorEvent** updates **EventUnits** and **EventString**. If an error is reported by a synchronous broadcast method, then **ErrorUnits** and **ErrorString** are set instead.*

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

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

When **AsyncMode** is false, then these methods operate synchronously and the Service returns to the application after completion. When operating synchronously, a **UposException** is raised if the method could not complete successfully.

- The Remote Order Display device may support transaction mode. A transaction is a sequence of display operations that are sent to a video unit as a single unit. Display operations which may be included in a transaction are **clearVideo**, **clearVideoRegion**, **copyVideoRegion**, **displayData**, **drawBox**, **restoreVideoRegion**, **saveVideoRegion**, and **updateVideoRegionAttribute**. During a transaction, the display operations are first validated. If valid, they are added to the transaction but not displayed yet. Once the application has added as many operations as required, then the transaction display method is called.

If the transaction is displayed synchronously, then any exception raised indicates that an error occurred during the display. If the transaction is displayed asynchronously, then the asynchronous display rules listed above are followed. If an error occurs and the **ErrorEvent** handler causes a retry, the entire transaction is retried.

Device Sharing

The Remote Order Display is an exclusive-use device. Its device sharing rules are:

- The application must claim the device before enabling it.
- The application must claim and enable the device before accessing many Remote Order Display specific properties.
- The application must claim and enable the device before calling methods that manipulate the device.
- When a **claim** method is called again, settable device characteristics are restored to their condition at **release**. Examples of restored characteristics are character set, video mode, and tone frequency. Region memory buffers, clock and cursor settings are considered state characteristics and are not restored.
- See the “Summary” table for precise usage prerequisites.

Properties (UML attributes)

AsyncMode Property

Syntax	AsyncMode: <i>int32</i> { read-write, access after open-claim-enable }
Remarks	If true, then the clearVideo , clearVideoRegion , copyVideoRegion , displayData , drawBox , restoreVideoRegion , saveVideoRegion , transactionDisplay , updateVideoRegionAttribute , and videoSound methods will be performed asynchronously. If false, they will be performed synchronously. This property is initialized to false by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

AutoToneDuration Property

Syntax	AsyncMode: <i>int32</i> { read-write, access after open-claim-enable }
Remarks	Holds the duration (in milliseconds) of the automatic tone for the video unit indicated in the CurrentUnitID property. This property is initialized to the default value for each online video unit when the device is first enabled following the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16. Some possible values of the exception’s <i>ErrorCode</i> property are:

Value	Meaning
E_ILLEGAL	An illegal value was specified. The ErrorString property is updated.

See Also **CurrentUnitID** Property.

AutoToneFrequency Property

Syntax	AutoToneFrequency: <i>int32</i> { read-write, access after open-claim-enable }
Remarks	Holds the frequency (in Hertz) of the automatic tone for the video unit indicated in the CurrentUnitID property. This property is initialized to the default value for each online video unit when the device is first enabled following the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16. Some possible values of the exception’s <i>ErrorCode</i> property are:

Value	Meaning
E_ILLEGAL	An illegal value was specified. The ErrorString property is updated.

See Also **CurrentUnitID** Property.

CapMapCharacterSet Property

Added in Release 1.7

Syntax	CapMapCharacterSet: <i>boolean</i> { read-only , access after open }
Remarks	<p>Defines the ability of the Service to map the characters of the application to the selected character set when displaying data.</p> <p>If CapMapCharacterSet is true, then the Service is able to map the characters to the character sets defined in CharacterSetList.</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 16.
See Also	CharacterSet Property, MapCharacterSet Property, CharacterSetList Property.

CapSelectCharacterSet Property

Syntax	CapSelectCharacterSet: <i>boolean</i> { read-only , access after open-claim-enable }
Remarks	<p>If true, the video unit indicated in the CurrentUnitID property may be loaded with an alternate, user supplied character set.</p> <p>This property is initialized for each video unit online when the device is first enabled following the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	CurrentUnitID Property.

CapTone Property

Syntax	CapTone: <i>boolean</i> { read-only , access after open-claim-enable }
Remarks	<p>If true, the video unit indicated in the CurrentUnitID property supports an enunciator.</p> <p>This property is initialized for each video unit online 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 16.
See Also	CurrentUnitID Property.

CapTouch Property

Syntax	CapTouch: <i>boolean</i> { read-only, access after open-claim-enable }
Remarks	If true, the video unit indicated in the CurrentUnitID property supports the ROD_DE_TOUCH_UP, ROD_DE_TOUCH_DOWN, and ROD_DE_TOUCH_MOVE event types. This property is initialized for each video unit online 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 16.
See Also	CurrentUnitID Property, DataEvent .

CapTransaction Property

Syntax	CapTransaction: <i>boolean</i> { read-only, access after open }
Remarks	If true, then transactions are supported by each video unit. 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 16.

CharacterSet Property

Updated in Release 1.5

Syntax	CharacterSet: <i>int32</i> { read-only, access after open-claim-enable }												
Remarks	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:												
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>Range 101 - 199</td> <td>Device-specific character sets that do not match a code page or the ASCII or ANSI character sets.</td> </tr> <tr> <td>Range 400 - 990</td> <td>Code page; matches one of the standard values.</td> </tr> <tr> <td>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> </tbody> </table>	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.												
	This property is initialized to the default video character set used by each video unit online when the device is first enabled following the open method. This is updated during the selectCharacterSet method.												
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.												
See Also	CurrentUnitID Property, CharacterSetList Property, CapSelectCharacterSet Property, selectCharacterSet method.												

CharacterSetList Property

Syntax	CharacterSetList: <i>string</i> { read-only, access after open-claim-enable }
Remarks	<p>Holds a string of character set numbers for the video unit indicated in the CurrentUnitID property.</p> <p>If CapSelectCharacterSet is true, this property is initialized for each video unit online when the device is first enabled following the open method.</p> <p>The character set number string consists of an ASCII numeric set of numbers, separated by commas.</p> <p>For example, if the string is “101, 850, 999”, the video unit supports a device-specific character set, code page 850, and the ANSI character set.</p>
Errors	A <code>UposException</code> may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	CurrentUnitID Property, CharacterSet Property, CapSelectCharacterSet Property, selectCharacterSet Method.

Clocks Property

Syntax	Clocks: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	<p>Holds the number of clocks the video unit, indicated in the CurrentUnitID property, can support.</p> <p>This property is initialized for each online video unit 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 16.
See Also	CurrentUnitID Property

CurrentUnitID Property

Syntax	CurrentUnitID: <i>int32</i> { read-write, access after open-claim-enable }				
Remarks	<p>Holds the current video unit ID. Up to 32 units are allowed on one Remote Order Display device. The unit ID definitions range from ROD_UID_1 to ROD_UID_32.</p> <p>The following properties and methods apply only to the selected video unit ID:</p> <ul style="list-style-type: none"> • Properties: AutoToneDuration, AutoToneFrequency, CapSelectCharacterSet, CapTone, CapTouch, CharacterSet, CharacterSetList, Clocks, VideoDataCount, VideoMode, VideoModesList, VideoSaveBuffers. <p>Setting CurrentUnitID will update these properties to the current values for the specified unit.</p> <p>Methods: checkHealth, clearInput, clearOutput.</p> <p>This property is initialized to ROD_UID_1 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 16.</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 illegal unit id was specified. The ErrorString property is updated.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	An illegal unit id was specified. The ErrorString property is updated.
Value	Meaning				
E_ILLEGAL	An illegal unit id was specified. The ErrorString property is updated.				

DataCount Property (Common)

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 VideoDataCount property.</p> <p>The application may read this property to determine whether additional input is enqueued from a device, but has not yet been delivered because of other application processing, freezing of events, or other causes.</p> <p>This property is initialized to zero by the open method.</p>
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	“Device Input Model” on page 18, VideoDataCount Property, DataEvent .

ErrorString Property

Syntax	ErrorString: <i>string</i> { read-only, access after open }
Remarks	<p>Holds a description of the error which occurred to the unit(s) specified by the ErrorUnits property, when an error occurs for any method that acts on a bitwise set of video units.</p> <p>If an error occurs during processing of an asynchronous request, the 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 16.
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 video 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 16.
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 16.
See Also	EventUnits Property, ErrorEvent .

EventType Property

Syntax	EventType: <i>int32</i> { read-write, access after open }				
Remarks	<p>Holds a bitwise mask that is used to selectively indicate which event types are to be delivered by the DataEvent, for all video units online. See the DataEvent description for event type definitions.</p> <p>This property is initialized to all defined event types by the open method.</p>				
Errors	<p>A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>E_ILLEGAL</td> <td>An illegal unit id was specified. The ErrorString property is updated.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	An illegal unit id was specified. The ErrorString property is updated.
Value	Meaning				
E_ILLEGAL	An illegal unit id was specified. The ErrorString property is updated.				
See Also	DataEvent.				

EventUnitID Property

Syntax	EventUnitID: <i>int32</i> { read-only, access after open-claim }
Remarks	Holds the video unit ID of the last delivered DataEvent . 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 16.
See Also	DataEvent.

EventUnits Property

Syntax	EventUnits: <i>int32</i> { read-only, access after open-claim }
Remarks	<p>Holds a bitwise mask of the unit(s) when an OutputCompleteEvent, output ErrorEvent, or StatusUpdateEvent is delivered.</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 16.
See Also	OutputCompleteEvent, ErrorEvent, StatusUpdateEvent.

MapCharacterSet Property

Added in Release 1.7

Syntax	MapCharacterSet: <i>boolean</i> { read-write, access after open }
Remarks	<p>If MapCharacterSet is true and when outputting data, the Service maps the characters transferred by the application to the character set selected in the CharacterSet property for displaying data.</p> <p>If MapCharacterSet is false, then no mapping is supported. In such a case the application has to ensure the mapping of the character set used in the application to the character set selected in the CharacterSet property.</p> <p>If CapMapCharacterSet is false, then this property is always 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 16.
See Also	CharacterSet Property, CapMapCharacterSet Property.

SystemClocks Property

Syntax	SystemClocks: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	<p>Holds the total number of clocks the Remote Order Display device can support at one time.</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 16.
See Also	Clocks Property.

SystemVideoSaveBuffers Property

Syntax	SystemVideoSaveBuffers: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	<p>Holds the total number of video save buffers the Remote Order Display device can support at one time.</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 16.
See Also	VideoSaveBuffers Property.

Timeout Property

- Syntax** **Timeout:** *int32* { read-write, access after open }
- Remarks** Holds the timeout value in milliseconds used by the Remote Order Display device to complete all output methods supported. If the device cannot successfully complete an output method within the timeout value, then the method throws a `UposException` if **AsyncMode** is false, or enqueues an **ErrorEvent** if **AsyncMode** is true.
- This property is initialized to a Service dependent default timeout following the **open** method.
- Errors** A `UposException` may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	An illegal unit id was specified. The ErrorString property is updated.

- See Also** **AsyncMode** Property.

UnitsOnline Property

- Syntax** **UnitsOnline:** *int32* { read-only, access after open-claim-enable }
- Remarks** Holds a bitwise mask indicating the video units online. Bit 0 is `ROD_UID_1`. 32 video units are supported. See “Model” on page 735.
- 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.
- Errors** A `UposException` may be thrown when this property is accessed. For further information, see “Errors” on page 16.

- See Also** “Model” on page 735, **checkHealth** Method, **StatusUpdateEvent**.

VideoDataCount Property

- Syntax** **VideoDataCount:** *int32* { read-only, access after open-claim-enable }
- Remarks** Holds the number of **DataEvents** enqueued for the video unit indicated in the **CurrentUnitID** property.
- The application may read this property to determine whether additional input is enqueued from a video unit, but has not yet been delivered because of other application processing, freeing of events, or other causes.
- 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 16.

- See Also** **CurrentUnitID** Property, **DataEvent**.

VideoMode Property

Syntax	VideoMode: <i>int32</i> { read-write, access after open-claim-enable }						
Remarks	<p>Holds the video ModeId selected for the video unit indicated by the CurrentUnitID property. The ModeId represents one of the selections in the VideoModesList property.</p> <p>This property is initialized to the Service dependent default video ModeId used by each video unit online 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 16.</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 illegal unit id was specified. The ErrorString property is updated.</td> </tr> <tr> <td>E_FAILURE</td> <td>An error occurred while communicating with the video unit indicated in the CurrentUnitID property. The ErrorString property is updated.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	An illegal unit id was specified. The ErrorString property is updated.	E_FAILURE	An error occurred while communicating with the video unit indicated in the CurrentUnitID property. The ErrorString property is updated.
Value	Meaning						
E_ILLEGAL	An illegal unit id was specified. The ErrorString property is updated.						
E_FAILURE	An error occurred while communicating with the video unit indicated in the CurrentUnitID property. The ErrorString property is updated.						
See Also	CurrentUnitID Property, VideoModesList Property.						

VideoModesList Property

Syntax	VideoModesList: <i>string</i> { read-only, access after open-claim-enable }
Remarks	<p>Holds the video modes supported for the video unit indicated in the CurrentUnitID property. The video modes are listed in a comma delineated string with the following format:</p> <p><ModeId>:<Height>x<Width>x<NumberOfColors><M C>.</p> <p>The ModeId values are determined by the Remote Order Display system. M = Monochrome (and gray scales) and C = Color.</p> <p>For example, if the string is “1:40x25x16C,2:80x25x16C”, then the video unit supports two video modes, ModeId 1 and ModeId 2. ModeId 1 has 40 rows, 25 columns, 16 colors, and is Color. ModeId 2 has 80 rows, 25 columns, 16 colors, and is Color.</p> <p>The ModeId is used to initialize the VideoMode property for each video unit online.</p> <p>This property is initialized to the video modes list supported by each video unit online when the device is first enabled following the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	CurrentUnitID Property, VideoMode Property.

VideoSaveBuffers Property

Syntax	VideoSaveBuffers: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	<p>Holds the number of save buffers for the video unit indicated in the CurrentUnitID property. This property should be consulted when using the saveVideoRegion, restoreVideoRegion and freeVideoRegion methods. When set to 0, this indicates that buffering for the selected unit is not supported. When this property is greater than 0, the Remote Order Display device can save at minimum one entire video screen for the selected video unit.</p> <p>This property is initialized for each video unit online when the device is first enabled following the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	CurrentUnitID Property, saveVideoRegion Method, restoreVideoRegion Method, freeVideoRegion Method.

Methods (UML operations)

checkHealth Method (Common)

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

The *level* parameter indicates the level 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 displayed on the video.
CH_INTERACTIVE	Perform an interactive test of the device. The 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 checks the health of the unit indicated in the **CurrentUnitID** property. If the current unit ID property is zero, an EROD_NOUNITS error is set. When the current unit ID property is set to a unit that is not currently online, the device will attempt to check the health of the video unit and report a communication error if necessary. The 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 UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_EXTENDED	<i>ErrorCodeExtended</i> = EROD_NOUNITS: The CurrentUnitID property is zero.
E_FAILURE	An error occurred while communicating with the video unit indicated in CurrentUnitID property.

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

clearInput Method (Common)

Syntax	clearInput (): void { raises-exception, use after open-claim }				
Remarks	Clears the device input that has been buffered for the unit indicated in the CurrentUnitID property. If the current unit ID property is zero, an EROD_NOUNITS is set. Any data events that are enqueued – usually waiting for DataEventEnabled to be set to true and FreezeEvents to be set to false – are also cleared.				
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16. Some possible values of the exception’s <i>ErrorCode</i> property are:				
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>E_EXTENDED</td> <td><i>ErrorCodeExtended</i> = EROD_NOUNITS: The CurrentUnitID property is zero.</td> </tr> </tbody> </table>	Value	Meaning	E_EXTENDED	<i>ErrorCodeExtended</i> = EROD_NOUNITS: The CurrentUnitID property is zero.
Value	Meaning				
E_EXTENDED	<i>ErrorCodeExtended</i> = EROD_NOUNITS: The CurrentUnitID property is zero.				
See Also	CurrentUnitID Property, “Device Input Model” on page 18.				

clearOutput Method (Common)**Updated in Release 1.7**

Syntax	clearOutput (): void { raises-exception, use after open-claim }				
Remarks	Clears all outputs that have been buffered, including all asynchronous output, for the unit indicated in the CurrentUnitID property, including video and tone outputs. If the current unit ID property is zero, an EROD_NOUNITS is set. Any output complete and output error events that are enqueued – usually waiting for DataEventEnabled to be set to true and FreezeEvents to be set to false – are also cleared.				
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16. Some possible values of the exception’s <i>ErrorCode</i> property are:				
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>E_EXTENDED</td> <td><i>ErrorCodeExtended</i> = EROD_NOUNITS: The CurrentUnitID property is set to zero.</td> </tr> </tbody> </table>	Value	Meaning	E_EXTENDED	<i>ErrorCodeExtended</i> = EROD_NOUNITS: The CurrentUnitID property is set to zero.
Value	Meaning				
E_EXTENDED	<i>ErrorCodeExtended</i> = EROD_NOUNITS: The CurrentUnitID property is set to zero.				
See Also	CurrentUnitID Property, “Device Output Models” on page 21.				

clearVideo Method

Syntax **clearVideo (units: *int32*, attribute: *int32*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
<i>attribute</i>	See Model on page 735 in the General Information section.

Remarks Clears the entire display area for the video unit(s) indicated in the *units* parameter. The display area will be cleared using the attribute placed in the *attribute* parameter.

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

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

See Also **AsyncMode** Property, “Model” on page 735

clearVideoRegion Method

Syntax **clearVideoRegion (units: *int32*, row: *int32*, column: *int32*, height: *int32*, width: *int32*, attribute: *int32*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
<i>row</i>	The region’s start row.
<i>column</i>	The region’s start column.
<i>height</i>	The number of rows in the region.
<i>width</i>	The number of columns in the region.
<i>attribute</i>	See “Model” on page 735 in the General Information section.

Remarks Clears the specified video region for the video unit(s) indicated in the *units* parameter. The display area will be cleared using the attribute placed in the *attribute* parameter.

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

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_FAILURE	An error occurred while communicating with one of the video units indicated in <i>units</i> . The ErrorUnits and ErrorString properties are updated. (Can only occur if AsyncMode is false.)

See Also **AsyncMode** Property, **ErrorString** Property, **ErrorUnits** Property, “Model” on page 735.

controlClock Method

Syntax **controlClock (units: *int32*, function: *int32*, clockId: *int32*, hour: *int32*, min: *int32*, sec: *int32*, row: *int32*, column: *int32*, attribute: *int32*, mode: *int32*);**
void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
<i>function</i>	The requested clock command. See values below.
<i>clockId</i>	Clock identification number. The valid values can be from 1 - Clocks . When the <i>function</i> parameter is ROD_CLK_PAUSE, ROD_CLK_RESUME, or ROD_CLK_STOP then <i>clockId</i> can be ROD_CLK_ALL to specify all clocks started on the specified video unit(s).
<i>hour</i>	The initial hours for the clock display.
<i>min</i>	The initial minutes for the clock display.
<i>sec</i>	The initial seconds for the clock display.
<i>row</i>	The clock's row.
<i>column</i>	The clock's start column.
<i>attribute</i>	See "Model" on page 735 in the General Information section.
<i>mode</i>	The type of clock to display. See values below.

The *function* parameter values are:

Value	Meaning
ROD_CLK_START	Starts a clock display assigned to the given <i>clockId</i> .
ROD_CLK_PAUSE	Temporarily stops a clock from updating the display until a ROD_CLK_RESUME requested.
ROD_CLK_RESUME	Resumes a clock that was previously paused, such that display updates continue.
ROD_CLK_STOP	Permanently stops the clock from updating the display and the <i>clockId</i> becomes free.
ROD_CLK_MOVE	Moves an instantiated clock to a new position.

The *mode* parameter values are:

Value	Meaning
ROD_CLK_SHORT	Displays a clock with "M:SS" format.
ROD_CLK_NORMAL	Displays a clock with "MM:SS" format.
ROD_CLK_12_int	Displays a 12 hour clock with "HH:MM:SS" format.
ROD_CLK_24_int	Displays a 24 hour clock with "HH:MM:SS" format.

Remarks Performs the clock command requested in the *function* parameter on the video unit(s) indicated in the *units* parameter. The clock will be displayed in the requested *mode* format at the location found in the *row* and *column* parameters. The clock will start at the specified *hour*, *min*, and *sec*, time values and will be updated every second until a ROD_CLK_PAUSE or ROD_CLK_STOP is requested for this *clockId*.

When a ROD_CLK_PAUSE, ROD_CLK_RESUME, or ROD_CLK_STOP command is issued, the *hour*, *min*, *sec*, *row*, *column*, *attribute*, and *mode* parameters are ignored. During a ROD_CLK_PAUSE command, the clock display updates are suspended. During a ROD_CLK_RESUME command, the clock updates continue.

If a ROD_CLK_PAUSE, ROD_CLK_RESUME, ROD_CLK_STOP or ROD_CLK_MOVE command is requested on an uninitialized *clockId* for any of the video units indicated in the *units* parameter, a EROD_BADCLK error is thrown. If a ROD_CLK_RESUME command is requested without doing a ROD_CLK_PAUSE, this has no effect and no exception is thrown.

When a ROD_CLK_MOVE command is issued, the clock is moved to the new location found in the *row* and *column* parameters. The *hour*, *min*, *sec*, *attribute* and *mode* parameters are ignored for this command function.

Generally a video unit can support the number of clocks indicated in the **Clocks** property. However, the ROD_CLK_START command will raise an exception containing EROD_NOCLOCKS if it exceeds the number of **SystemClocks** even though the **Clocks** property may indicate the unit can support more clocks than allocated for that unit.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_EXTENDED	<p><i>ErrorCodeExtended</i> = EROD_BADCLK: A ROD_CLK_PAUSE, ROD_CLK_RESUME, ROD_CLK_START, ROD_CLK_MOVE command was requested and the specified <i>clockId</i> has not been initialized by the ROD_CLK_START command.</p> <p><i>ErrorCodeExtended</i> = EROD_NOCLOCKS: The ROD_CLK_START failed because the number of SystemClocks has been reached.</p> <p>The ErrorUnits and ErrorString properties are updated.</p>
E_FAILURE	An error occurred while communicating with one of the video units indicated in the <i>units</i> parameter. The ErrorUnits and ErrorString properties are updated.
E_BUSY	When a ROD_CLK_START command is requested but the specified <i>clockId</i> is in use. The ErrorUnits and ErrorString properties are updated.

See Also **Clocks** Property, **ErrorString** Property, **ErrorUnits** Property, “Model” on page 735.

controlCursor Method

Syntax **controlCursor (units: *int32*, function: *int32*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
-----------	-------------

<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
--------------	--

<i>function</i>	The cursor command, indicating the type of cursor to display. See values below.
-----------------	---

Value	Meaning
-------	---------

ROD_CRS_LINE	enable a solid underscore line.
--------------	---------------------------------

ROD_CRS_LINE_BLINK	enable a blinking solid underscore cursor.
--------------------	--

ROD_CRS_BLOCK	enable a solid block cursor.
---------------	------------------------------

ROD_CRS_BLOCK_BLINK	enable a blinking solid block cursor.
---------------------	---------------------------------------

ROD_CRS_OFF	Disable cursor.
-------------	-----------------

Remarks Enables or disables the cursor depending on the *function* parameter, for the video unit(s) indicated in the *units* parameter.

When the *function* is ROD_CRS_OFF, the cursor is disabled, otherwise the cursor is enabled as the requested cursor type. If the video unit cannot support the requested cursor type, the Service will use the next closest cursor type.

The cursor attribute is taken from the current cursor location.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
-------	---------

E_FAILURE	An error occurred communicating with one of the video units indicated in <i>units</i> . The ErrorUnits and ErrorString properties are updated.
-----------	--

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

copyVideoRegion Method

Syntax **copyVideoRegion (units: *int32*, row: *int32*, column: *int32*, height: *int32*, width: *int32*, targetRow: *int32*, targetColumn: *int32*): void { raises-exception, use after open-claim-enable }**

Parameter	Description
<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
<i>row</i>	The region's start row.
<i>column</i>	The region's start column.
<i>height</i>	The number of rows in the region.
<i>width</i>	The number of columns in the region.
<i>targetRow</i>	The start row of the target location.
<i>targetColumn</i>	The start column of the target location.

Remarks Copies a region of the display area to a new location on the display area for the video unit(s) indicated in the *units* parameter. The source area is defined by the *row*, *column*, *height*, and *width* parameters. The top-left corner of the target location is defined by the *targetRow* and *targetColumn* parameters. If the ranges overlap the copy is done such that all original data is preserved.

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

Errors A UposException may be thrown when this method is invoked. For further information, see "Errors" on page 16.

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

Value	Meaning
E_FAILURE	An error occurred while communicating with one of the video units indicated in <i>units</i> . The ErrorUnits and ErrorString properties are updated. (Can only occur if AsyncMode is false.)

See Also **AsyncMode** Property, **ErrorString** Property, **ErrorUnits** Property, "Model" on page 735.

drawBox Method

Syntax **drawBox (units: *int32*, row: *int32*, column: *int32*, height: *int32*, width: *int32*, attribute: *int32*, bordertype: *int32*):**
void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
<i>row</i>	The box's start row.
<i>column</i>	The box's start column.
<i>height</i>	The number of rows in the box.
<i>width</i>	The number of columns in the box.
<i>attribute</i>	The video attribute. See "Model" on page 735 in the General Information section.
<i>bordertype</i>	The border type to be drawn. Can be any printable character or a defined border type. See values below.
Value	Meaning

ROD_BDR_SINGLE	A single line border.
ROD_BDR_DOUBLE	A double line border.
ROD_BDR_SOLID	A solid block border.

Remarks Draws a box on the video unit(s) indicated in the *units* parameter.

The Remote Order Display will attempt to draw a box with the border type specified. If the character set does not support the chosen border type, the Service will choose the best fit from the given character set.

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

Errors A UposException may be thrown when this method is invoked. For further information, see "Errors" on page 16.

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

Value	Meaning
E_FAILURE	An error occurred while communicating with one of the video units indicated in <i>units</i> . The ErrorUnits and ErrorString properties are updated.

See Also **AsyncMode** Property, **ErrorString** Property, **ErrorUnits** Property, "Model" on page 735.

freeVideoRegion Method

Syntax **freeVideoRegion (units: *int32*, bufferId: *int32*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
<i>bufferId</i>	Number identifying the video buffer to free. Valid values range from 1 to the VideoSaveBuffers property for a selected unit(s).

Remarks Frees any buffer memory allocated for the video unit(s) indicated in the *units* parameter. The number of video buffers supported is stored in the **VideoSaveBuffers** property for each video unit online. If the *bufferId* was never used in a previous **saveVideoRegion** method, no action is taken.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_FAILURE	An error occurred while communicating with one of the video units indicated in <i>units</i> . The ErrorUnits and ErrorString properties are updated.

See Also **ErrorString** Property, **ErrorUnits** Property, **VideoSaveBuffers** Property, **saveVideoRegion** Method.

resetVideo Method

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

units is a bitwise mask indicating which video unit(s) to operate on.

Remarks Sets the video unit(s) indicated in the *units* parameter to a power on state. All Service buffers and clocks associated with the unit(s) are released. All settable characteristics are set to default values.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_FAILURE	An error occurred while communicating with one of the video units indicated in <i>units</i> . The ErrorUnits and ErrorString properties are updated.

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

restoreVideoRegion Method

Syntax **restoreVideoRegion (units: *int32*, targetRow: *int32*, targetColumn: *int32*,
bufferId: *int32*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
<i>targetRow</i>	The start row of the target location.
<i>targetColumn</i>	The start column of the target location.
<i>bufferId</i>	Number identifying the source video buffer to use. Valid values range from 1 to the VideoSaveBuffers property for the selected unit(s).

Remarks Restores a previously saved video region of the display area from the requested *bufferId* for the video unit(s) indicated in the *units* parameter. A region can be saved using the **saveVideoRegion** method. The number of video buffers supported is stored in the **VideoSaveBuffers** property for each video unit online. The target location is defined by the *targetRow* and *targetColumn* parameters. This method doesn't free the memory after restoring, therefore, this method can be used to copy a video region to multiple locations on the display. Use the **freeVideoRegion** method to free any memory allocated for a video buffer.

If the *bufferId* does not contain a previously saved video region for the *units* selected, a EROD_NOREGION exception is raised.

Video regions cannot be restored between video units. For example, the **saveVideoRegion** method is called with *units* = 0000 1000 and *bufferId* = 1. This will save a video region for the Unit Id 4, in to Buffer 1 for that unit. If this method is called with *units* = 0000 0100 and *bufferId* = 1 with the intention of restoring the previously saved buffer to Unit Id 3, then either a UposException with *ErrorCode* of EROD_NOREGION would be thrown, or an unwanted region would be restored.

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

Errors A UposException may be thrown when this method is invoked. For further information, see "Errors" on page 16.

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

Value	Meaning
E_EXTENDED	<i>ErrorCodeExtended</i> = EROD_NOREGION: The <i>bufferId</i> does not contain a previously saved video region.
E_FAILURE	An error occurred while communicating with one of the video units indicated in <i>units</i> . The ErrorUnits and ErrorString properties are updated. (Can only occur if AsyncMode is false.)

See Also **AsyncMode** Property, **ErrorString** Property, **ErrorUnits** Property, **VideoSaveBuffers** Property, **saveVideoRegion** Method.

saveVideoRegion Method

Syntax **saveVideoRegion (units: *int32*, row: *int32*, column: *int32*, height: *int32*, width: *int32*, bufferId: *int32*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
<i>row</i>	The start row of the region to save.
<i>column</i>	The start column of the region to save.
<i>height</i>	The number of rows in the region to save.
<i>width</i>	The number of columns in the region to save.
<i>bufferId</i>	Number identifying the video buffer to use. Valid values range from 1 to the VideoSaveBuffers property for a selected unit(s).

Remarks Saves the specified video region of the display area to one of the provided video buffers for the video unit(s) indicated in the *units* parameter. The number of video buffers supported is stored in the **VideoSaveBuffers** property for each video unit online. However, a **UposException** will be raised if the requested buffer exceeds the number of **SystemVideoSaveBuffers** even though the **VideoSaveBuffers** property may indicated the unit can support more save buffers than currently allocated for that unit.

If **VideoSaveBuffers** is greater than 0, the Service will be able to support at minimum one entire video screen. This does not guarantee that the Service can save an entire video screen in each supported buffer for a single unit. A **UposException** is raised when all the buffer memory has been allocated for a specific unit.

The source area is defined by the *row*, *column*, *height*, and *width* parameters. The video region can be restored to the screen by calling the **restoreVideoRegion** method. If **saveVideoRegion** is called twice with the same *bufferId*, the previous video data is lost, and any allocated memory is returned to the system.

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

Errors A **UposException** may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	<i>bufferId</i> , <i>row</i> , <i>column</i> , <i>height</i> , or <i>width</i> is out of range. The ErrorUnits and ErrorString properties are updated.
E_EXTENDED	<i>ErrorCodeExtended</i> = EROD_NOBUFFERS: Requested buffer exceeds the number of SystemVideoSaveBuffers .

ErrorCodeExtended = EROD_NOROOM:

All the buffer memory has been allocated for a specific unit. The **ErrorUnits** and **ErrorString** properties are updated.

E_FAILURE

An error occurred while communicating with one of the video units indicated in *units*. The **ErrorUnits** and **ErrorString** properties are updated. (Can only occur if **AsyncMode** is false.)

See Also **AsyncMode** Property, **ErrorString** Property, **ErrorUnits** Property, **SystemVideoSaveBuffers** Property, **VideoSaveBuffers** Property, **restoreVideoRegion** Method.

selectCharacterSet Method

Syntax `selectCharacterSet (units: int32, characterSet: int32):
void { raises-exception, use after open-claim-enable }`

Parameter	Description
<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
<i>characterSet</i>	Contain the character set for displaying characters. Values are:
Value	Meaning
Range 101 - 199	A device-specific character set that does not match a code page, nor the ASCII or ANSI character sets.
Range 400 - 990	Code page; matches one of the standard values.
ROD_CS_UNICODE	The character set supports UNICODE. The value of this constant is 997.
ROD_CS_ASCII	The ASCII character set, supporting the ASCII characters between 20-hex and 7F-hex. The value of this constant is 998.
ROD_CS_ANSI	The ANSI character set. The value of this constant is 999.

Remarks Selects a compatible character set for the video unit(s) indicated in the *units* parameter.

The **CharacterSet** property is updated for each video unit id that is successfully assigned a new character set.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_FAILURE	An error occurred while communicating with one of the video units indicated in <i>units</i> . The ErrorUnits and ErrorString properties are updated.

See Also **ErrorString** Property, **ErrorUnits** Property, **CapSelectCharacterSet** Property, **CharacterSet** Property.

setCursor Method

Syntax **setCursor (units: *int32*, row: *int32*, column: *int32*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
<i>row</i>	Row to place the cursor on.
<i>column</i>	Column to place the cursor on.

Remarks Updates the cursor position on the video unit(s) indicated in the *units* parameter.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_FAILURE	An error occurred while communicating with one of the video units indicated in <i>units</i> . The ErrorUnits and ErrorString properties are updated.

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

transactionDisplay Method

Syntax **transactionDisplay (units: *int32*, function: *int32*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
<i>function</i>	Transaction control function. Valid values are:

Value	Meaning
ROD_TD_TRANSACTION	Begin a transaction.
ROD_TD_NORMAL	End a transaction by displaying the buffered data.

Remarks Enters or exits transaction mode for the video unit(s) indicated in the *units* parameter.

If *function* is ROD_TD_TRANSACTION, then transaction mode is entered. Subsequent calls to **clearVideo**, **clearVideoRegion**, **copyVideoRegion**, **displayData**, **drawBox**, **restoreVideoRegion**, **saveVideoRegion**, and **updateVideoRegionAttribute** will buffer the display data (either at the video unit or the Service, depending on the display capabilities) until **transactionDisplay** is called with the *function* parameter set to ROD_TD_NORMAL. (In this case, the display methods only validate the method parameters and buffer the data – they do not initiate displaying. Also, the value of the **AsyncMode** property does not affect their operation: No **OutputID** will be assigned to the request, nor will an **OutputCompleteEvent** be enqueued.)

If *function* is ROD_TD_NORMAL, then transaction mode is exited. If some data was buffered by calls to the methods **clearVideo**, **clearVideoRegion**, **copyVideoRegion**, **displayData**, **drawBox**, **restoreVideoRegion**, **saveVideoRegion**, and **updateVideoRegionAttribute**, then the buffered data is displayed. The entire transaction is treated as one message. This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

Calling the **clearOutput** method cancels transaction mode for the unit indicated in the **CurrentUnitID** property. Any buffered print lines are also cleared.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_BUSY	Cannot perform while output is in progress for one of the video units indicated in <i>units</i> . The ErrorUnits and ErrorString properties are updated. (Can only occur if AsyncMode is false and function is ROD_TD_NORMAL.)
E_FAILURE	An error occurred while communicating with one of the video units indicated in <i>units</i> . The ErrorUnits and ErrorString properties are updated. (Can only occur if AsyncMode is false and function is ROD_TD_NORMAL.)

See Also **clearVideo** Method, **clearVideoRegion** Method, **copyVideoRegion** Method, **displayData** Method, **drawBox** Method, **restoreVideoRegion** Method, **saveVideoRegion** Method, **updateVideoRegionAttribute** Method.

updateVideoRegionAttribute Method

Syntax **updateVideoRegionAttribute (units: int32, function: int32, row: int32, column: int32, height: int32, width: int32, attribute: int32): void { raises-exception, use after open-claim-enable }**

Parameter	Description
<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
<i>function</i>	The attribute command. See values below.
<i>row</i>	The region’s start row.
<i>column</i>	The region’s start column.
<i>height</i>	The number of rows in the region.
<i>width</i>	The number of columns in the region.
<i>attribute</i>	See Model on page 735 in the General Information section.

The *function* parameter values are:

Value	Meaning
ROD_UA_SET	Set the region with the new attribute.
ROD_UA_INTENSITY_ON	Turn on foreground intensity in the region.
ROD_UA_INTENSITY_OFF	Turn off foreground intensity in the region.
ROD_UA_REVERSE_ON	Reverse video the region.
ROD_UA_REVERSE_OFF	Remove reverse video from the region.
ROD_UA_BLINK_ON	Turn on blinking in the region.
ROD_UA_BLINK_OFF	Turn off blinking in the region.

Remarks Modifies the attribute on the video unit(s) indicated in the *units* parameter in the region defined by the *row*, *column*, *height*, and *width* parameters. When the *function* parameter is ROD_UA_SET, the region's attributes will be replaced with the new value in the *attribute* parameter; otherwise the *attribute* parameter is ignored and the region's attributes will be modified.

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

Errors A UposException may be thrown when this method is invoked. For further information, see "Errors" on page 16.

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

Value	Meaning
E_FAILURE	An error occurred while communicating with one of the video units indicated in <i>units</i> . The ErrorUnits and ErrorString properties are updated. (Can only occur if AsyncMode is false.)

See Also **AsyncMode** Property, **ErrorString** Property, **ErrorUnits** Property, "Model" on page 735.

videoSound Method

Syntax **videoSound (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 video unit(s) to operate on.
<i>frequency</i>	Tone frequency in Hertz.
<i>duration</i>	Tone duration in milliseconds.
<i>numberOfCycles</i>	If UPOS_FOREVER, then start tone sounding and, repeat continuously. Else perform the specified number of cycles.
<i>interSoundWait</i>	When <i>numberOfCycles</i> is not one, then pause for <i>interSoundWait</i> milliseconds before repeating the tone cycle (before playing the tone again).

Remarks Sounds the video enunciator for the video(s) indicated in the *units* parameter.

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

The duration of a video tone cycle is:

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

After the video has started an asynchronous sound, then the **clearOutput** method will stop the sound. (When an *interSoundWait* value of UPOS_FOREVER was used to start the sound, then the application must use **clearOutput** to stop the continuous sounding of tones.)

If **CapTone** is false for the selected unit(s), a UposException is raised.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_FAILURE	An error occurred while communicating with one of the video units indicated in <i>units</i> . The ErrorUnits and ErrorString properties are updated. (Can only occur if AsyncMode is false.)

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

Events (UML interfaces)

DataEvent

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

Description Notifies the application when input data from a video touch unit is available.

Attributes This event contains the following attribute:

Attribute	Type	Description
-----------	------	-------------

<i>Status</i>	<i>int32</i>	As described below
---------------	--------------	--------------------

The *Status* attribute is divided into four bytes as indicated below:

High Word		Low Word (Event Type)
High Byte	Low Byte	ROD_DE_TOUCH_UP ROD_DE_TOUCH_DOWN ROD_DE_TOUCH_MOVE
Row	Column	

The low word contains the Event type. The high word contains additional data depending on the Event type. When the Event type is ROD_DE_TOUCH_UP, ROD_DE_TOUCH_DOWN, or ROD_DE_TOUCH_MOVE, the high word indicates where the touch occurred. The low byte contains the Column position and the high byte contains the Row position, with valid values ranging from 0-255.

Remarks This event can be filtered at the Remote Order Display device by setting the **EventType** property.

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

See Also “Device Input Model” on page 18, **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 Remote Order Display 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 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 Remote Order Display devices which may not have any knowledge of the Service's need for this event.

See Also "Events" on page 15, **directIO** Method.

ErrorEvent

Updated in Release 1.7

<< 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 Remote Order Display error has been detected and a suitable response by the application 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 list of <i>ErrorCodes</i> on page 16.
<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.

ErrorResponse *int32* 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 processing 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 all buffered output data (including all 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 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 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 21, “Device Information Reporting Model” on page 26, **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 property 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 Enqueued when a previously started asynchronous output request completes successfully. The **EventUnits** property is updated before the event is delivered.

See Also **EventUnits** Property, “Device Output Models” on page 21.

StatusUpdateEvent

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

Description Notifies the application that there is a change in the power status of a video unit.

Attributes This event contains the following attribute:

Attribute	Type	Description
<i>Status</i>	<i>int32</i>	Reports a change in the power state of a display. <i>Note that Release 1.3</i> added Power State Reporting with additional <i>Power reporting StatusUpdateEvent values</i> . See “StatusUpdateEvent” description on page 63.

Remarks Enqueued when the Remote Order Display detects a power state change.
 Deviation from the standard **StatusUpdateEvent** (see page 63):

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

See Also **EventUnits** Property.

Scale

This Chapter defines the Scale device category.

Summary

Properties (UML attributes)

<i>Common</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
AutoDisable:	<i>boolean</i>	{ read-write }	1.3	open
CapPowerReporting:	<i>int32</i>	{ read-only }	1.3	open
CapStatisticsReporting:	<i>boolean</i>	{ read-only }	1.8	open
CapUpdateStatistics:	<i>boolean</i>	{ read-only }	1.8	open
CheckHealthText:	<i>string</i>	{ read-only }	1.0	open
Claimed:	<i>boolean</i>	{ read-only }	1.0	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 }	1.0	open & claim
FreezeEvents:	<i>boolean</i>	{ read-write }	1.0	open
OutputID:	<i>int32</i>	{ read-only }	1.0	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 }	1.0	--
DeviceControlDescription:	<i>string</i>	{ read-only }	1.0	--
DeviceControlVersion:	<i>int32</i>	{ read-only }	1.0	--
DeviceServiceDescription:	<i>string</i>	{ read-only }	1.0	open
DeviceServiceVersion:	<i>int32</i>	{ read-only }	1.0	open
PhysicalDeviceDescription:	<i>string</i>	{ read-only }	1.0	open
PhysicalDeviceName:	<i>string</i>	{ read-only }	1.0	open

Properties (Continued)

<i>Specific</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
CapDisplay:	<i>boolean</i>	{ read-only }	1.2	open
CapDisplayText:	<i>boolean</i>	{ read-only }	1.3	open
CapPriceCalculating:	<i>boolean</i>	{ read-only }	1.3	open
CapTareWeight:	<i>boolean</i>	{ read-only }	1.3	open
CapZeroScale:	<i>boolean</i>	{ read-only }	1.3	open
AsyncMode:	<i>boolean</i>	{ read-write }	1.3	open
MaxDisplayTextChars:	<i>int32</i>	{ read-only }	1.3	open
MaximumWeight:	<i>int32</i>	{ read-only }	1.0	open
SalesPrice:	<i>currency</i>	{ read-only }	1.3	open, claim, & enable
TareWeight:	<i>int32</i>	{ read-write }	1.3	open, claim, & enable
UnitPrice:	<i>currency</i>	{ read-write }	1.3	open, claim, & enable
WeightUnit:	<i>int32</i>	{ read-only }	1.0	open

Methods (UML operations)**Common**

<i>Name</i>	<i>Version</i>
open (logicalDeviceName: <i>string</i>): void { raises-exception }	1.3
close (): void { raises-exception, use after open }	1.3
claim (timeout: <i>int32</i>): void { raises-exception, use after open }	1.3
release (): void { raises-exception, use after open, claim }	1.3
checkHealth (level: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3
clearInput (): void { raises-exception, use after open, claim }	1.3
clearOutput (): void { }	<i>Not supported</i>
directIO (command: <i>int32</i>, inout data: <i>int32</i>, inout obj: <i>object</i>): void { raises-exception, use after open }	1.3
resetStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
retrieveStatistics (inout statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
updateStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8

Specific***Name***

displayText (data: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.3
readWeight (inout weightData: <i>int32</i>, timeout: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.3
zeroScale (): void { raises-exception, use after open, claim, enable }	1.3

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>
upos::events::DataEvent			1.3
Status:	<i>int32</i>	{ read-only }	
upos::events::DirectIOEvent			1.0
EventNumber:	<i>int32</i>	{ read-only }	
Data:	<i>int32</i>	{ read-write }	
Obj:	<i>object</i>	{ read-write }	
upos::events::ErrorEvent			1.3
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		<i>Not Supported</i>	
upos::events::StatusUpdateEvent			1.3
Status:	<i>int32</i>	{ read-only }	

General Information

The Scale programmatic name is “Scale”.

Capabilities

The scale Device has the following capability:

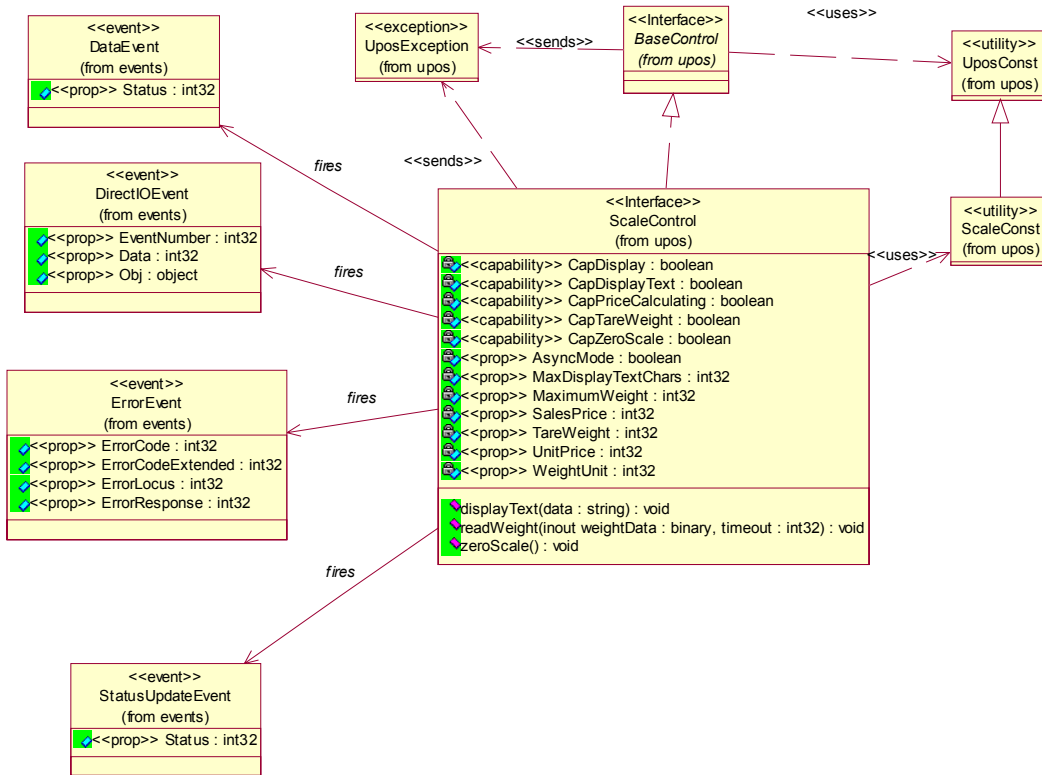
- Provides item weight to the application. The measure of weight may be in grams, kilograms, ounces, or pounds, depending upon the scale device.

The scale may have the following additional capabilities:

- Includes an integrated display with the current weight, or with the current weight plus application-specified text.
- Performs price calculations (weight X unit price) and returns the sale price. (This feature is mostly used in Europe at this time.)
- Supports application setting of tare weight.
- Supports application zeroing of the scale.

Scale Class Diagram

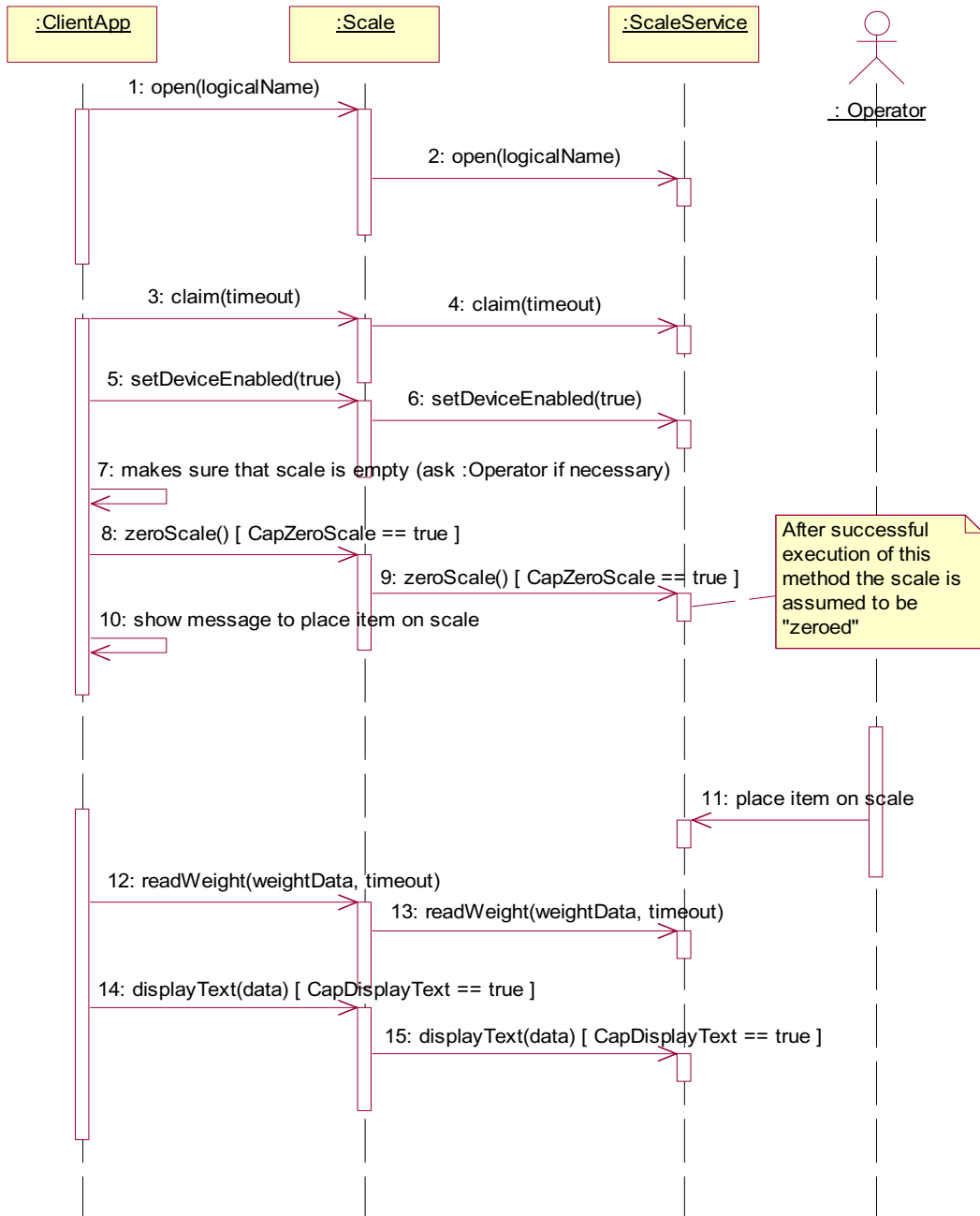
The following diagram shows the relationships between the Scale classes.



Scale Sequence Diagram *Added in Release 1.7*

The following sequence diagram shows the typical synchronous usage of a Scale device.

NOTE: we are assuming that the :ClientApp already successfully opened and enabled the Scale device.



Model

The general model of a scale is:

- A scale returns the weight of an item placed on its weighing surface.
- The primary scale method is **readWeight**. By default, it is performed synchronously. It returns after reading data from the scale; the weight is returned in the **readWeight**'s *weightData* parameter. If an error occurs or if the timeout elapses, a *UposException* will be thrown.

- ***UnifiedPOS Release 1.3 and later - Asynchronous Input***

If the **AsyncMode** property is true when **readWeight** is called, then the method is performed asynchronously. It initiates event driven input and returns immediately. The *timeout* parameter specifies the maximum time the application wants to wait for a settled weight. Additional points are:

- If an error occurs while initiating event driven input (such as the device is offline), then a *UposException* is thrown. Otherwise, **readWeight** returns immediately to the application, and scale processing continues asynchronously.
- If a settled weight is received, then a **DataEvent** is enqueued containing the weight data in the *Status* property.
- If a scale error occurs (including a timeout with no settled weight), then an **ErrorEvent** is enqueued. The application event handler may retry the weighing process by setting the event's *ErrorResponse* property to *ER_RETRY*.
- Only one asynchronous call to **readWeight** can be in progress at a time. An attempt to nest asynchronous scale operations will result in a *UposException* being thrown.
- An asynchronous scale operation may be cancelled with the **clearInput** method.

For price-calculating scales, the application should set the **UnitPrice** property before calling **readWeight**. After a weight is read (and just before the **DataEvent** is delivered to the application, for asynchronous mode), the **SalesPrice** property is set to the calculated price of the item.

Device Sharing

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

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

Properties (UML attributes)

AsyncMode Property

Added in Release 1.3

Syntax	AsyncMode: <i>boolean</i> { read-write, access after open }
Remarks	If true, then the readWeight method will be performed asynchronously. If false, the readWeight method will be performed synchronously. This property is initialized to false by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	readWeight Method.

CapDisplay Property

Syntax	CapDisplay: <i>boolean</i> { read-only, access after open }
Remarks	If true, the scale includes an integrated display that shows the current weight. If false, the application may need to show the current weight on another display. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	CapDisplayText Property, MaxDisplayTextChars Property.

CapDisplayText Property

Added in Release 1.3

Syntax	CapDisplayText: <i>boolean</i> { read-only, access after open }
Remarks	If true, the scale includes an integrated display that shows the current weight and can also show text that describes the item being weighed. If false, extra text cannot be shown on the display. If true, then CapDisplay must also be true. 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 16.
See Also	CapDisplay Property, MaxDisplayTextChars Property.

CapPriceCalculating Property***Added in Release 1.3***

Syntax	CapPriceCalculating: <i>boolean</i> { read-only, access after open }
Remarks	If true, the scale can calculate prices. If false, the scale only returns a weight. For price calculating scales the calculation unit is in the scale rather than in the data-receiving terminal. 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 16.
See Also	readWeight Method, WeightUnit Property, UnitPrice Property, SalesPrice Property.

CapTareWeight Property***Added in Release 1.3***

Syntax	CapTareWeight: <i>boolean</i> { read-only, access after open }
Remarks	If true, the scale includes setting a tare value. If false, the scale does not support tare 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 16.
See Also	TareWeight Property.

CapZeroScale Property***Added in Release 1.3***

Syntax	CapZeroScale: <i>boolean</i> { read-only, access after open }
Remarks	If true, the application can set the scale weight to zero. If false, the scale does not support programmatic zeroing. 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 16.
See Also	zeroScale Method.

MaxDisplayTextChars Property***Added in Release 1.3***

Syntax	MaxDisplayTextChars: <i>int32</i> { read-only, access after open }
Remarks	<p>Holds the number of characters that may be displayed on an integrated display for the text which describes an article.</p> <p>If CapDisplayText is false, then the device does not support text displaying and this property is always zero.</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 16.
See Also	CapDisplay Property, CapDisplayText Property.

MaximumWeight Property

Syntax	MaximumWeight: <i>int32</i> { read-only, access after open }
Remarks	<p>Holds the maximum weight measurement possible from the scale. The measurement unit is available via the WeightUnit property.</p> <p>This property has an assumed decimal place located after the “thousands” digit position. For example, an actual value of 12345 represents 12.345, and an actual value of 5 represents 0.005.</p> <p>Changing the WeightUnit property will also cause this property to change.</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 16.
See Also	WeightUnit Property.

SalesPrice Property **Added in Release 1.3/Updated in Release 1.6**

Syntax	SalesPrice: <i>currency</i> { read-only, access after open }
Remarks	<p>Holds the sales price read from the scale for price calculating scales. For price calculating scales the scale calculates this value during the process of weighing by multiplying the UnitPrice property by the acquired weight. This property is a monetary value stored using an implied four decimal places. For example, an actual value of 12345 represents 1.2345.</p> <p>This property is set before the readWeight method returns (in synchronous mode) or the DataEvent is delivered (in asynchronous mode).</p> <p>If CapPriceCalculating is false, then the device is not a price calculating scale and SalesPrice is always zero.</p> <p>This property is initialized to zero when the device is first enabled following the open method. (In releases prior to 1.5, this description stated that initialization took place by the open method. In Release 1.5, it was updated for consistency with other devices.)</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	readWeight Method, WeightUnit Property, CapPriceCalculating Property, UnitPrice Property.

TareWeight Property **Added in Release 1.3/Updated in Release 1.6**

Syntax	TareWeight: <i>int32</i> { read-write, access after open }				
Remarks	<p>Holds the tare weight of scale data. This property has an assumed decimal place located after the “thousands” digit position. For example, an actual value of 12345 represents 12.345, and an actual value of 5 represents 0.005. The measured unit is specified in the WeightUnit property. If CapTareWeight is false, then the device does not support setting of a tare value and this property is always zero.</p> <p>Tare weight is not included in the item weight returned by the readWeight method.</p> <p>This property is initialized to the scale’s default tare weight (usually zero), when the device is first enabled following the open method. (In releases prior to 1.5, this description stated that initialization took place by the open method. In Release 1.5, it was updated for consistency with other devices.)</p>				
Errors	<p>A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Value</th> <th style="text-align: left;">Meaning</th> </tr> </thead> <tbody> <tr> <td>E_ILLEGAL</td> <td>CapTareWeight is false or an invalid tare value was specified.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	CapTareWeight is false or an invalid tare value was specified.
Value	Meaning				
E_ILLEGAL	CapTareWeight is false or an invalid tare value was specified.				
See Also	readWeight Method, WeightUnit Property, CapTareWeight Property.				

UnitPrice Property**Added in Release 1.3/Updated in Release 1.6****Syntax** **UnitPrice: *currency* { read-write, access after open }****Remarks** Holds the unit price of the article to be weighed. For price calculating scales this property is to be set before calling the **readWeight** method. During weighing, the scale sets the **SalesPrice** property to the product of the item's weight and this property. This property is a monetary value stored using an implied four decimal places. For example, an actual value of 12345 represents 1.2345.

If **CapPriceCalculating** is false, then setting of a unit price is not supported and this property is always zero.

This property is initialized to zero when the device is first enabled following the **open** method. (In releases prior to 1.5, this description stated that initialization took place by the **open** method. In Release 1.5, it was updated for consistency with other devices.)

Errors A UposException may be thrown when this property is accessed. For further information, see "Errors" on page 16.

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

Value	Meaning
E_ILLEGAL	CapPriceCalculating is false or an invalid price was specified.

See Also **readWeight** Method, **WeightUnit** Property, **CapPriceCalculating** Property, **SalesPrice** Property.**WeightUnit Property****Syntax** **WeightUnit: *int32* { read-only, access after open }****Remarks** Holds the unit of weight of scale data, and has one of the following values:

Value	Meaning
SCAL_WU_GRAM	Unit is a gram.
SCAL_WU_KILOGRAM	Unit is a kilogram (= 1000 grams).
SCAL_WU_OUNCE	Unit is an ounce.
SCAL_WU_POUND	Unit is a pound (= 16 ounces).

This property is initialized to the scale's weight unit by the **open** method.

Errors A UposException may be thrown when this property is accessed. For further information, see "Errors" on page 16.

Methods (UML operations)

displayText Method

Updated in Release 1.7

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

Parameter	Description
-----------	-------------

<i>data</i> ¹	The string of characters to display.
--------------------------	--------------------------------------

Remarks If **CapDisplayText** is true, updates the text shown on the integrated display. Calling this method with an empty string (“”) will clear the display.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
-------	---------

E_ILLEGAL	An invalid text was specified -- the text contains more characters than MaxDisplayTextChars , or CapDisplayText is false.
-----------	---

See Also **CapDisplay** Property, **CapDisplayText** Property, **MaxDisplayTextChars** Property.

¹. In the **OPOS** environment, the format of this data depends upon the value of the **BinaryConversion** property. See **BinaryConversion** property on page A-28.

readWeight Method

Syntax **readWeight (inout weightData: *int32*, timeout: *int32*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>weightData</i>	If AsyncMode is false, contains the returned value for the weight measured by the scale, else zero.
<i>timeout</i>	The number of milliseconds to wait for a settled weight before failing the method. If zero, the method attempts to read the scale weight, then returns the appropriate status immediately. If UPOS_FOREVER (-1), the method waits as long as needed until a weight is successfully read or an error occurs.

Remarks Reads a weight from the scale.

The weight returned, *weightData*, has an assumed decimal place located after the “thousands” digit position. For example, an actual value of 12345 represents 12.345, and an actual value of 5 represents 0.005.

Release 1.2

The weighing process is performed synchronously and the method will return after finishing the weighing process. The weight is returned in the *weightData* parameter.

Release 1.3 and later

If **AsyncMode** is false, then **readWeight** operates synchronously, as with earlier releases.

If **AsyncMode** is true, the weighing process is performed asynchronously. The method will initiate a read, then return immediately. Once the weighing process is complete, a **DataEvent** is delivered with the item’s weight contained in the event’s *Status* property.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	An invalid <i>timeout</i> parameter was specified.
E_TIMEOUT	A stable non-zero weight was not available before <i>timeout</i> milliseconds elapsed (only if AsyncMode is false).
E_EXTENDED	<i>ErrorCodeExtended</i> = ESCAL_OVERWEIGHT: The weight was over MaximumWeight (can only be returned if AsyncMode is false).

See Also **UnitPrice** Property, **WeightUnit** Property, **CapPriceCalculating** Property, **SalesPrice** Property, **TareWeight** Property.

zeroScale Method***Added in Release 1.3***

Syntax	zeroScale (): void { raises-exception, use after open-claim-enable }				
Remarks	If CapZeroScale is true, sets the current scale weight to zero. It may be used for initial calibration, or to account for tare weight on the scale.				
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16. Some possible values of the exception’s <i>ErrorCode</i> property are: <table><thead><tr><th>Value</th><th>Meaning</th></tr></thead><tbody><tr><td>E_ILLEGAL</td><td>CapZeroScale is false.</td></tr></tbody></table>	Value	Meaning	E_ILLEGAL	CapZeroScale is false.
Value	Meaning				
E_ILLEGAL	CapZeroScale is false.				
See Also	CapZeroScale Property.				

Events (UML interfaces)

DataEvent

Added in Release 1.3

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

Description Notifies the application that an asynchronous **readWeight** has completed.

Attributes This event contains the following attribute:

Attribute	Type	Description
<i>Status</i>	<i>int32</i>	The weight of the item.

Remarks If the scale is a price calculating scale, the unit price is placed in the **UnitPrice** property and the calculated sales price is placed in the **SalesPrice** property before this event is delivered.

See Also “Events” on page 15.

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 Scale 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 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 Scale devices which may not have any knowledge of the Service’s need for this event.

See Also “Events” on page 15, **directIO** Method.

ErrorEvent**Added in Release 1.3**

```

<< 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 scale device error has been detected and a suitable response by the application 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 list of <i>ErrorCodes</i> on page 16.
<i>ErrorCodeExtended</i>	<i>int32</i>	Extended error code causing the error event. 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 has one of the following values:

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_RETRY	Retry the asynchronous input. The error state is exited.
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 **DataEventEnabled** 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 scale data. This event is not delivered until **DataEventEnabled** is true, so that proper application sequencing occurs.

See Also “Events” on page 15.

StatusUpdateEvent

Added in Release 1.3

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

Description Notifies the application that a scale has had an operation status change.

Attributes This event contains the following attribute:

Attribute	Type	Description
<i>Status</i>	<i>int32</i>	Indicates a status change.

Note that Release 1.3 added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See “StatusUpdateEvent” description on page 63.

Remarks Enqueued when a change in status of the scale device has occurred.

See Also “Events” on page 15.

Scanner (Bar Code Reader)

This Chapter defines the Scanner device category.

Summary

Properties (UML attributes)

<i>Common</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
AutoDisable:	<i>boolean</i>	{ read-write }	1.2	open
CapPowerReporting:	<i>int32</i>	{ read-only }	1.3	open
CapStatisticsReporting:	<i>boolean</i>	{ read-only }	1.8	open
CapUpdateStatistics:	<i>boolean</i>	{ read-only }	1.8	open
CheckHealthText:	<i>string</i>	{ read-only }	1.0	open
Claimed:	<i>boolean</i>	{ read-only }	1.0	open
DataCount:	<i>int32</i>	{ read-only }	1.2	open
DataEventEnabled:	<i>boolean</i>	{ read-write }	1.0	open
DeviceEnabled:	<i>boolean</i>	{ read-write }	1.0	open & claim
FreezeEvents:	<i>boolean</i>	{ read-write }	1.0	open
OutputID:	<i>int32</i>	{ read-only }	1.0	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 }	1.0	--
DeviceControlDescription:	<i>string</i>	{ read-only }	1.0	--
DeviceControlVersion:	<i>int32</i>	{ read-only }	1.0	--
DeviceServiceDescription:	<i>string</i>	{ read-only }	1.0	open
DeviceServiceVersion:	<i>int32</i>	{ read-only }	1.0	open
PhysicalDeviceDescription:	<i>string</i>	{ read-only }	1.0	open
PhysicalDeviceName:	<i>string</i>	{ read-only }	1.0	open

Properties (Continued)

<i>Specific</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
DecodeData:	<i>boolean</i>	{ read-write }	1.2	open
ScanData:	<i>binary</i>	{ read-only }	1.0	open
ScanDataLabel:	<i>binary</i>	{ read-only }	1.2	open
ScanDataType:	<i>int32</i>	{ read-only }	1.2	open

Methods (UML operations)**Common**

<i>Name</i>	<i>Version</i>
open (logicalDeviceName: <i>string</i>): void { raises-exception }	1.0
close (): void { raises-exception, use after open }	1.0
claim (timeout: <i>int32</i>): void { raises-exception, use after open }	1.0
release (): void { raises-exception, use after open, claim }	1.0
checkHealth (level: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.0
clearInput (): void { raises-exception, use after open, claim }	1.0
clearOutput (): void { }	<i>Not supported</i>
directIO (command: <i>int32</i>, inout data: <i>int32</i>, inout obj: <i>object</i>): void { raises-exception, use after open }	1.0
resetStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
retrieveStatistics (inout statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
updateStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8

Specific*None*

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>
upos::events::DataEvent			1.0
Status:	<i>int32</i>	{ read-only }	
upos::events::DirectIOEvent			1.0
EventNumber:	<i>int32</i>	{ read-only }	
Data:	<i>int32</i>	{ read-write }	
Obj:	<i>object</i>	{ read-write }	
upos::events::ErrorEvent			1.0
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		<i>Not Supported</i>	
upos::events::StatusUpdateEvent			1.3
Status:	<i>int32</i>	{ read-only }	

General Information

The Scanner programmatic name is “Scanner”.

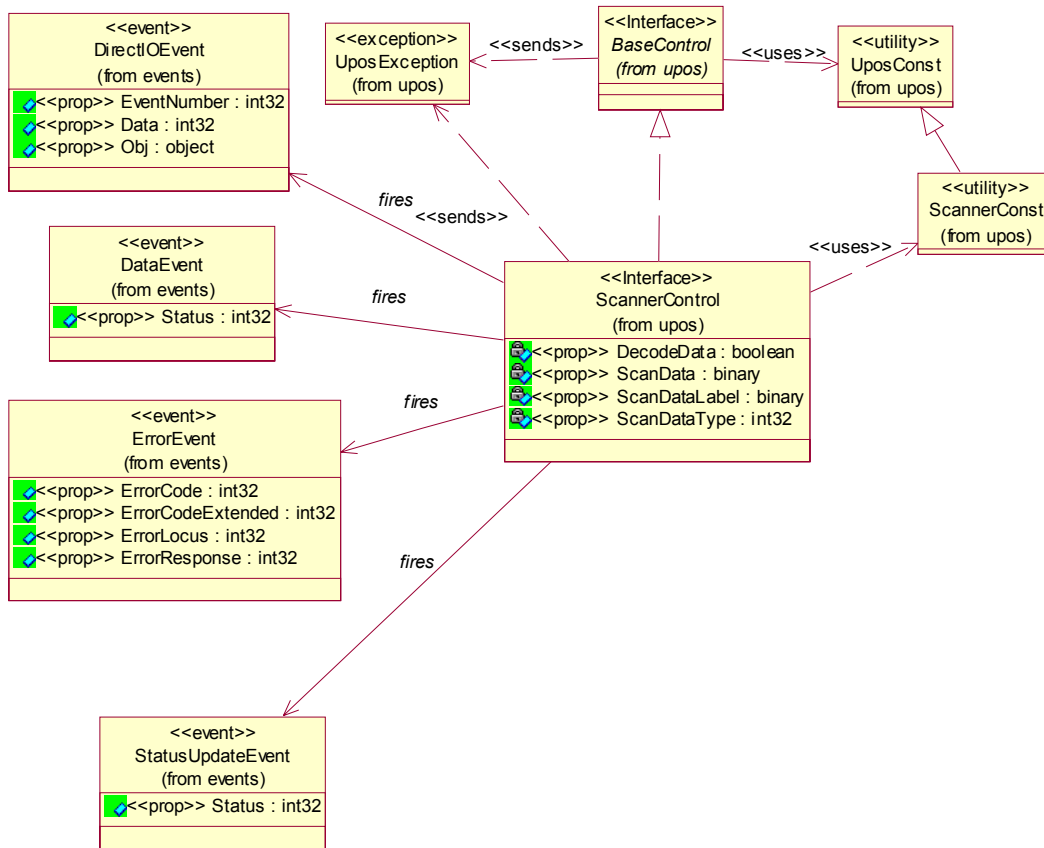
Capabilities

The Scanner Device has the following capability:

- Reads encoded data from a label.

Scanner Class Diagram

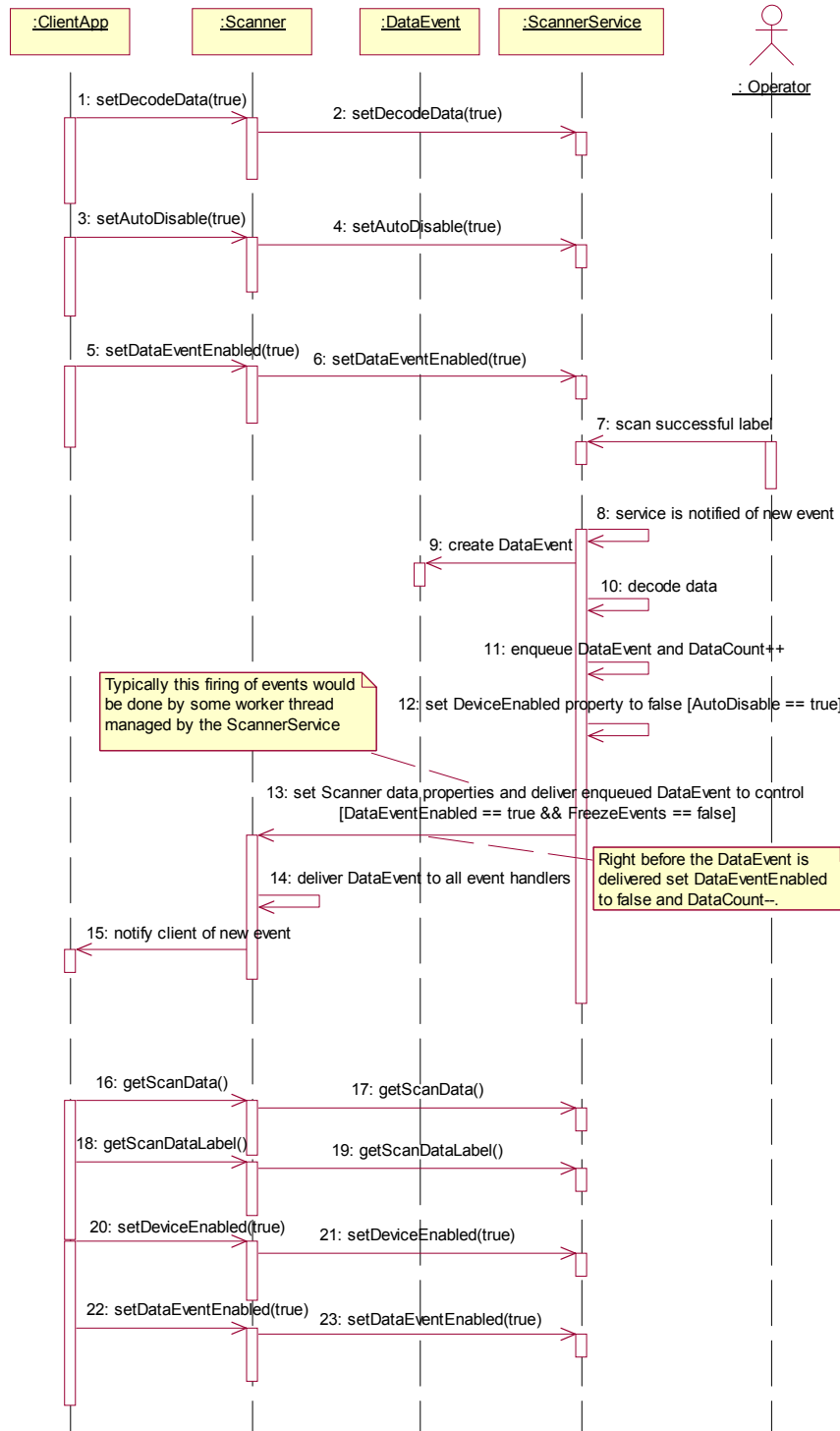
The following diagram shows the relationships between the Scanner classes.



Scanner Sequence Diagram Updated in Release 1.8

The following sequence diagram shows the typical usage of a Scanner device.

NOTE: we are assuming that the :ClientApp already successfully registered event handlers and opened, claimed and enabled the Scanner device. This means that the Claimed, DeviceEnabled properties are == true



Model

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

- When input is received from the scanner, a **DataEvent** is enqueued.
- If the **AutoDisable** property is true, then the device automatically disables 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 **DataEventEnabled** to false. This causes subsequent input data to be enqueued while the application processes the current input and associated properties. When the application has finished processing the current input and is ready for more data, it reenables events by setting **DataEventEnabled** to true.
- An **ErrorEvent** (or events) is enqueued if an error occurs while gathering or processing input, and is delivered to the application when **DataEventEnabled** is true and other event delivery requirements are met.
- The **DataCount** property may be read to obtain the total number of enqueued **DataEvents**.
- All enqueued input may be deleted by calling **clearInput**. See the **clearInput** method description for more details.

Scanned data is placed into the property **ScanData**. If the application sets the property **DecodeData** to true, then the data is decoded into the **ScanDataLabel** and **ScanDataType** properties.

Device Sharing

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

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

Properties (UML attributes)

DecodeData Property

Syntax	DecodeData: <i>boolean</i> { read-write, access after open }
Remarks	If true, then ScanData will be decoded into the properties ScanDataLabel and ScanDataType . 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 16.
See Also	“Device Input Model” on page 18

ScanData Property**Updated in Release 1.7****Syntax** **ScanData: *binary* { read-only, access after open } ¹****Remarks** Holds the data read from the scanner.

Scan data is, in general, in the format as delivered from the scanner. Message header and trailer information are removed, however, since they do not contain useful information for an application and are likely to be scanner-specific.

Common header information is a prefix character (such as an STX character). Common trailer information is a terminator character (such as an ETX or CR character) and a block check character if one is generated by the scanner.

This property should include a symbology character if one is returned by the scanner (for example, an ‘A’ for UPC-A). It should also include check digits if they are present in the label and returned by the scanner. (Note that both symbology characters and check digits may or may not be present, depending upon the scanner configuration. The scanner will return them if present, but will not generate or calculate them if they are absent.)

Some merchandise may be marked with a supplemental barcode. This barcode is typically placed to the right of the main barcode, and consists of an additional two or five characters of information. If the scanner reads merchandise that contains both main and supplemental barcodes, the supplemental characters are appended to the main characters, and the result is delivered to the application as one label. (Note that a scanner may support configuration that enables or disables the reading of supplemental codes.)

Some merchandise may be marked with multiple labels, sometimes called multi-symbol labels or tiered labels. These barcodes are typically arranged vertically, and may be of the same or different symbology. If the scanner reads merchandise that contains multiple labels, each barcode is delivered to the application as a separate label. This is necessary due to the current lack of standardization of these barcode types. One is not able to determine all variations based upon the individual barcode data. Therefore, the application will need to determine when a multiple label barcode has been read based upon the data returned. (Note that a scanner may or may not support reading of multiple labels.)

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

Errors A `UposException` may be thrown when this property is accessed. For further information, see “Errors” on page 16.**See Also** “Device Input Model” on page 18

¹. In the **OPOS** environment, the format of this data depends upon the value of the **BinaryConversion** property. See **BinaryConversion** property on page A-28.

ScanDataLabel Property**Updated in Release 1.8****Syntax** **ScanDataLabel: *binary* { read-only, access after open }²****Remarks** Holds the decoded bar code label.

When **DecodeData** is false, this property will have zero length. When **DecodeData** is true, then **ScanData** is decoded into this property as follows:

- Scanner-generated symbology characters are removed, if present.
- If a label type's human readable text contains a printed check digit (such as with UPC-A and EAN-13), then the check digit must be present in this property. If the scanner does not return the check digit to the Service, then it is to be calculated and appended to form the complete label.
- For variable length bar codes, the length identification is removed, if present.

For example, the EAN-13 barcode which appears printed as "5 018374 827715" on a label may be received from the scanner and placed into **ScanData** as the following:

Received from scanner	ScanData	Comment
5018374827715	5018374827715	Complete barcode only
501837482771<CR>	501837482771	Without check digit with carriage return
F5018374827715<CR>	F5018374827715	With scanner-dependent symbology character and carriage return
<STX>F5018374827715<ETX>	F5018374827715	With header, symbology character, and trailer

For each of these cases (and any other variations), this property must always be set to the string "5018374827715", and **ScanDataType** must be set to SCAN_SDT_EAN13.

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

Errors A UposException may be thrown when this property is accessed. For further information, see "Errors" on page 16.

See Also "Device Input Model" on page 18

². In the **OPOS** environment, the format of this data depends upon the value of the **BinaryConversion** property. See **BinaryConversion** property on page A-28.

ScanDataType Property**Updated in Release 1.8****Syntax** ScanDataType: *int32* { read-only, access after open }**Remarks** Holds the decoded bar code label type.

When **DecodeData** is false, this property is set to SCAN_SDT_UNKNOWN.
 When **DecodeData** is true, the Service tries to determine the scan label type. The following label types are defined:

Value	Label Type
<i>One Dimensional Symbologies</i>	
SCAN_SDT_UPCA	UPC-A
SCAN_SDT_UPCA_S	UPC-A with supplemental barcode
SCAN_SDT_UPCE	UPC-E
SCAN_SDT_UPCE_S	UPC-E with supplemental barcode
SCAN_SDT_UPCD1	UPC-D1
SCAN_SDT_UPCD2	UPC-D2
SCAN_SDT_UPCD3	UPC-D3
SCAN_SDT_UPCD4	UPC-D4
SCAN_SDT_UPCD5	UPC-D5
SCAN_SDT_EAN8	EAN 8 (= JAN 8)
SCAN_SDT_JAN8	JAN 8 (= EAN 8)
SCAN_SDT_EAN8_S	EAN 8 with supplemental barcode
SCAN_SDT_EAN13	EAN 13 (= JAN 13)
SCAN_SDT_JAN13	JAN 13 (= EAN 13)
SCAN_SDT_EAN13_S	EAN 13 with supplemental barcode
SCAN_SDT_EAN128	EAN-128
SCAN_SDT_TF	Standard (or discrete) 2 of 5
SCAN_SDT_ITF	Interleaved 2 of 5
SCAN_SDT_Codabar	Codabar
SCAN_SDT_Code39	Code 39
SCAN_SDT_Code93	Code 93
SCAN_SDT_Code128	Code 128
SCAN_SDT_OCRA	OCR "A"
SCAN_SDT_OCRB	OCR "B"

Value	Label Type
<i>One Dimensional Symbologies - Added in Release 1.8</i>	
SCAN_SDT_RSS14	14 digit GTIN only
SCAN_SDT_RSS_EXPANDED	14 digit GTIN plus additional defined fields (e.g., price, weight)

Value	Label Type
<i>Composite Symbologies - Added in Release 1.8</i>	
SCAN_SDT_CCA	Composite Component A. Up to 56 characters of data.
SCAN_SDT_CCB	Composite Component B. Up to 338 characters of data.
SCAN_SDT_CCC	Composite Component C. Up to 2361 characters of data.

A Composite Component may occur with any one of several different label types, such as UPC, EAN, and RSS. The composite component is read at the same time as the linear component. When such a label is read, a **DataEvent** is delivered that sets **ScanDataType** to SCAN_SDT_CCA, SCAN_SDT_CCB, or SCAN_SDT_CCC. The next **DataEvent** always delivers the linear component. (In other words, the Service enqueues two **DataEvents** at the same time: First the composite component, then the linear component.) It is the application writer's responsibility to merge the data associated with the two **DataEvents**.

Value	Label Type
<i>Two Dimensional Symbologies</i>	
SCAN_SDT_PDF417	PDF 417
SCAN_SDT_MAXICODE	MAXICODE

Value	Label Type
<i>Special Cases</i>	
SCAN_SDT_OTHER	If greater or equal to this type, then the Service has returned an undefined symbology.
SCAN_SDT_UNKNOWN	The Service cannot determine the barcode symbology. ScanDataLabel may not be properly formatted for the actual barcode type.

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

Errors A `UposException` may be thrown when this property is accessed. For further information, see “Errors” on page 16.

See Also “Device Input Model” on page 18

Events (UML interfaces)

DataEvent

<< event >> **upos::events::DataEvent**

Status: *int32* { read-only }

Description Notifies the application that input data from the Scanner (Bar Code Reader) is available.

Attributes This event contains the following attribute:

Attribute	Type	Description
-----------	------	-------------

<i>Status</i>	<i>int32</i>	Always zero.
---------------	--------------	--------------

Remarks The scanner data is placed in the **ScanData**, **ScanDataLabel**, and **ScanDataType** properties prior to a **DataEvent** being delivered to the application.

See Also “Events” on page 15

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 Scanner 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 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 Scanner devices which may not have any knowledge of the Service’s need for this event.

See Also “Events” on page 15, **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 scanner device error has been detected and a suitable response by the application 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 list of <i>ErrorCodes</i> on page 16.
<i>ErrorCodeExtended</i>	<i>int32</i>	Extended error code causing the error event. 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 has one of the following values:

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 DataEventEnabled 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 scanner data. This event is not delivered until **DataEventEnabled** is true, so that proper application sequencing occurs.

See Also "Events" on page 15

StatusUpdateEvent

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

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

Attributes This event contains the following attribute:

Attribute	Type	Description
<i>Status</i>	<i>int32</i>	Reports a change in the power state of a Scanner device.

Note that Release 1.3 added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See "StatusUpdateEvent" description on page 63.

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

See Also "Events" on page 15

Signature Capture

This Chapter defines the Signature Capture device category.

Summary

Properties (UML attributes)

<i>Common</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
AutoDisable:	<i>boolean</i>	{ read-write }	1.2	open
CapPowerReporting:	<i>int32</i>	{ read-only }	1.3	open
CapStatisticsReporting:	<i>boolean</i>	{ read-only }	1.8	open
CapUpdateStatistics:	<i>boolean</i>	{ read-only }	1.8	open
CheckHealthText:	<i>string</i>	{ read-only }	1.0	open
Claimed:	<i>boolean</i>	{ read-only }	1.0	open
DataCount:	<i>int32</i>	{ read-only }	1.2	open
DataEventEnabled:	<i>boolean</i>	{ read-write }	1.0	open
DeviceEnabled:	<i>boolean</i>	{ read-write }	1.0	open & claim
FreezeEvents:	<i>boolean</i>	{ read-write }	1.0	open
OutputID:	<i>int32</i>	{ read-only }	1.0	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 }	1.0	--
DeviceControlDescription:	<i>string</i>	{ read-only }	1.0	--
DeviceControlVersion:	<i>int32</i>	{ read-only }	1.0	--
DeviceServiceDescription:	<i>string</i>	{ read-only }	1.0	open
DeviceServiceVersion:	<i>int32</i>	{ read-only }	1.0	open
PhysicalDeviceDescription:	<i>string</i>	{ read-only }	1.0	open
PhysicalDeviceName:	<i>string</i>	{ read-only }	1.0	open

Properties (Continued)

<i>Specific</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
CapDisplay:	<i>boolean</i>	{ read-only }	1.0	open
CapRealTimeData:	<i>boolean</i>	{ read-only }	1.2	open
CapUserTerminated:	<i>boolean</i>	{ read-only }	1.0	open
MaximumX:	<i>int32</i>	{ read-only }	1.0	open
MaximumY:	<i>int32</i>	{ read-only }	1.0	open
PointArray:	<i>array of points</i>	{ read-only }	1.0	open, claim, & enable
RawData:	<i>binary</i>	{ read-only }	1.0	open, claim, & enable
RealTimeDataEnabled:	<i>boolean</i>	{ read-write }	1.2	open

Methods (UML operations)**Common**

<i>Name</i>	<i>Version</i>
open (logicalDeviceName: <i>string</i>): void { raises-exception }	1.0
close (): void { raises-exception, use after open }	1.0
claim (timeout: <i>int32</i>): void { raises-exception, use after open }	1.0
release (): void { raises-exception, use after open, claim }	1.0
checkHealth (level: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.0
clearInput (): void { raises-exception, use after open, claim }	1.0
clearOutput (): void { }	<i>Not supported</i>
directIO (command: <i>int32</i>, inout data: <i>int32</i>, inout obj: <i>object</i>): void { raises-exception, use after open }	1.0
resetStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
retrieveStatistics (inout statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
updateStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8

Specific***Name***

beginCapture (formName: string): void { raises-exception, use after open, claim, enable }	1.0
endCapture (): void { raises-exception, use after open, claim, enable }	1.0

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>
upos::events::DataEvent			1.0
Status:	<i>int32</i>	{ read-only }	
upos::events::DirectIOEvent			1.0
EventNumber:	<i>int32</i>	{ read-only }	
Data:	<i>int32</i>	{ read-write }	
Obj:	<i>object</i>	{ read-write }	
upos::events::ErrorEvent			1.0
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		<i>Not Supported</i>	
upos::events::StatusUpdateEvent			1.3
Status:	<i>int32</i>	{ read-only }	

General Information

The Signature Capture programmatic name is “SignatureCapture”.

Capabilities

The Signature Capture Device has the following capability:

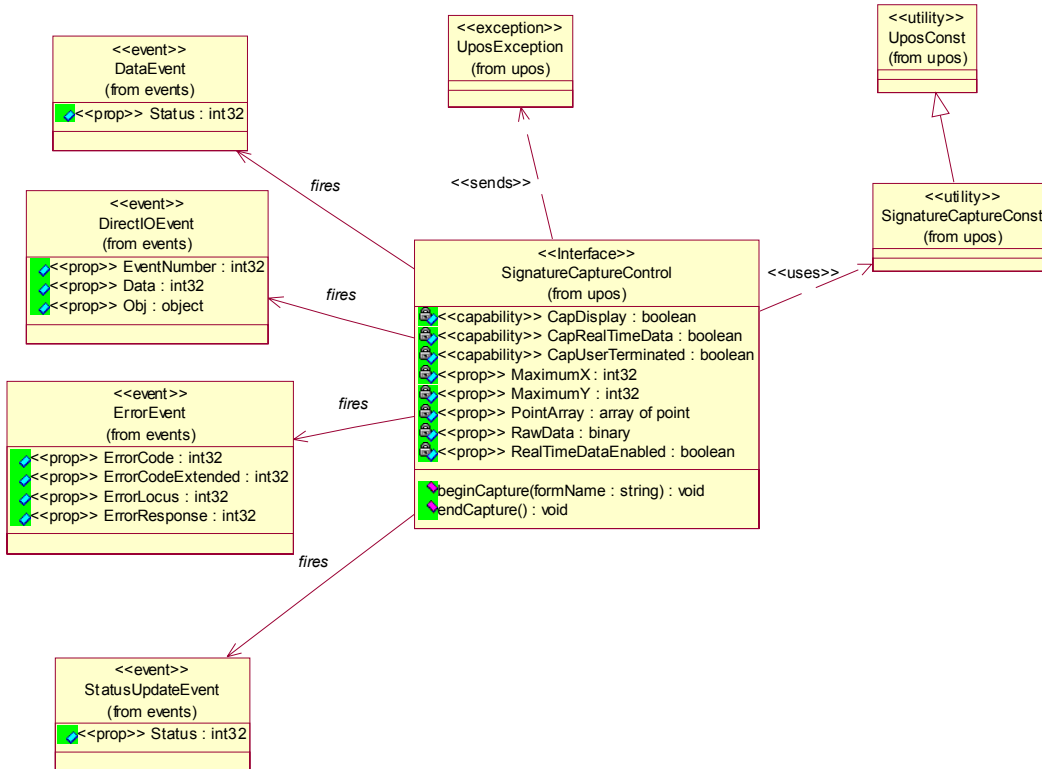
- Obtains a signature captured by a signature capture device. The captured signature data is in the form of lines consisting of a series of points. Each point lies within the co-ordinate system defined by the resolution of the device, where (0, 0) is the upper-left point of the device, and (**MaximumX**, **MaximumY**) is the lower-right point. The signature line points are presented to the application by a **DataEvent** with a single array of line points

The Signature Capture Device may have the following additional capabilities:

- Provides a way for the user to terminate signature capture – that is, to tell the device that she or he has completed the signature.
- Displays form/data on the signature capture device.
- Returns the signature in “real time” as it is entered on the device. If this capability is true and has been enabled by application by setting the **RealTimeDataEnabled** property to true, then a series of **DataEvents** are enqueued, each with an array of one or more line points representing a partial signature.

Signature Capture Class Diagram

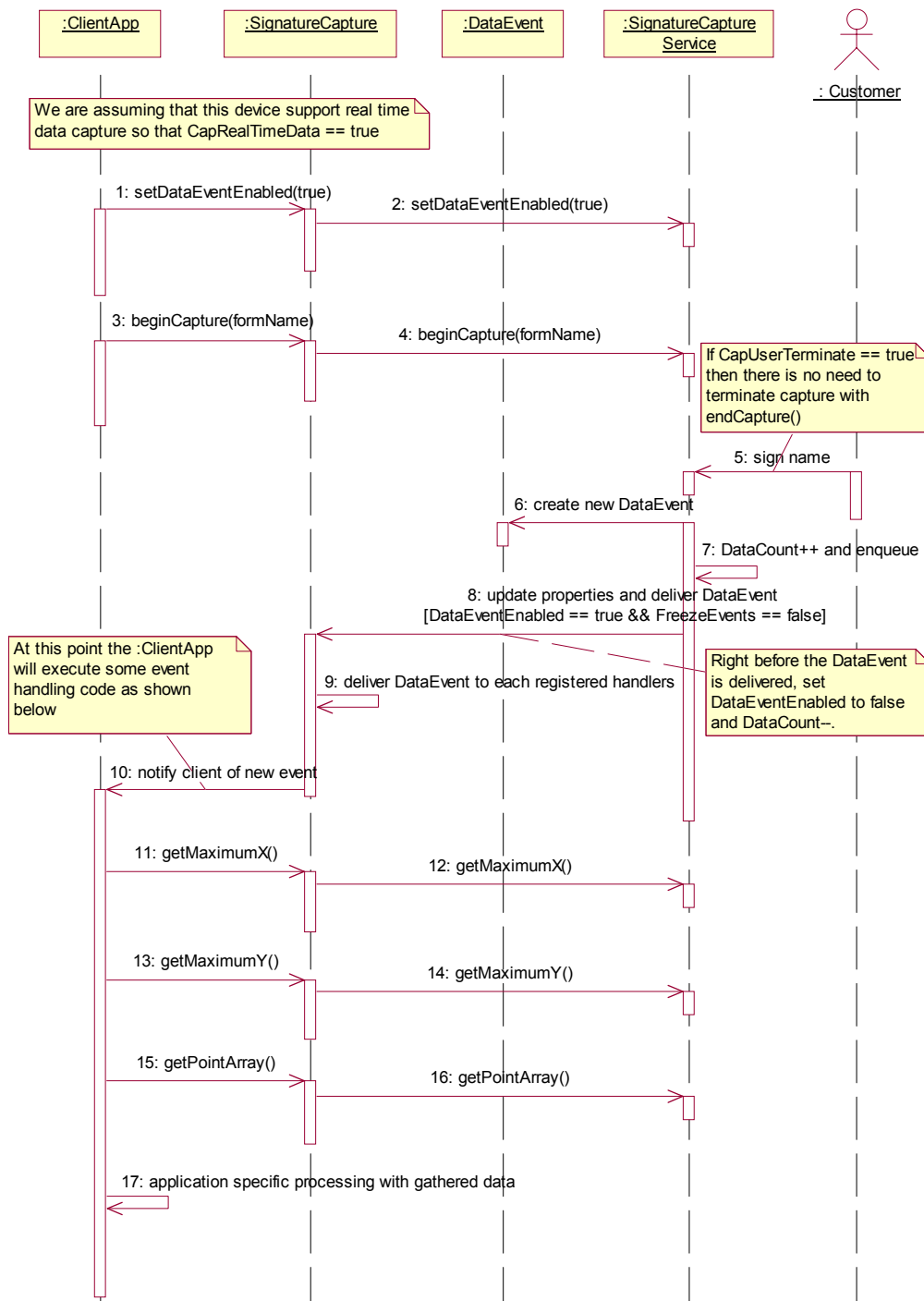
The following diagram shows the relationships between the Signature Capture classes.



Signature Capture Sequence Diagram *Updated in Release 1.8*

The following sequence diagram shows the typical usage of gathering data from a Signature Capture device.

NOTE: we are assuming that the :ClientApp already successfully registered event handlers and opened, claimed and enabled the SignatureCapture device. This means that the Claimed, DeviceEnabled properties are == true



Model

The signature capture device usage model is:

- Open and claim the device.
- Enable the device and set the property **DataEventEnabled** to true.
- Begin capturing a signature by calling **beginCapture**. This method displays a form or data screen (if the device has a display) and enables the stylus.
- If the device is capable of supplying signature data in real time as the signature is entered (**CapRealTimeData** is true), and if **RealTimeDataEnabled** is true, the signature is presented to the application as a series of partial signature data events until the signature capture is terminated.
- If the device provides a way for the user to terminate the signature, then when the user terminates, a **DataEvent** is enqueued. Otherwise, the application must call **endCapture** to terminate the signature.
- Disable the device. If the device has a display, this also clears the display.

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

- When input is received by the Service, it enqueues a **DataEvent**.
- If **AutoDisable** is true, then the Device automatically disables itself when a **DataEvent** is enqueued. However, note that setting **AutoDisable** probably is not very useful for the Signature Capture control. If **RealTimeDataEnabled** is true, then **AutoDisable** does not make sense. If **RealTimeDataEnabled** is false, then the pacing of signatures is controlled by the application via the **beginCapture** method. It is probably in the best interests of the application not to use the **AutoDisable** property for this device class.
- A queued **DataEvent** can be delivered to the application when the property **DataEventEnabled** is true and other event delivery requirements are met. Just before delivering this event, data is copied into properties, and further data events are disabled by setting **DataEventEnabled** to false. This causes subsequent input data to be enqueued while the application processes the current input and associated properties. When the application has finished processing the current input and is ready for more data, it re-enables events by setting **DataEventEnabled** to true.
- An **ErrorEvent** (or events) is enqueued if the an error occurs while gathering or processing input, and is delivered to the application when **DataEventEnabled** is true and other event delivery requirements are met.
- The **DataCount** property may be read to obtain the number of queued **DataEvents**.
- All enqueued input may be deleted by calling **clearInput**. See the **clearInput** method description for more details.

Deviations from the general “Device Input Model” for event-driven input are:

- The capture of signature data begins when **beginCapture** is called.
- If signature capture is terminated by calling **endCapture**, then no **DataEvent** will be enqueued.

Device Sharing

The Signature Capture is an exclusive-use device, as follows:

- The application must claim the device before enabling it.
- The application must claim and enable the device before calling methods that manipulate the device or before changing some writable properties.
- See the “Summary” table for precise usage prerequisites.

Properties (UML attributes)

CapDisplay Property

Syntax	CapDisplay: <i>boolean</i> { read-only, access after open }
Remarks	If true, the device is able to display a form or data entry screen. 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 16.

CapRealTimeData Property

Syntax	CapRealTimeData: <i>boolean</i> { read-only, access after open }
Remarks	If true, the device is able to supply signature data as the signature is being captured (“real time”). This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

CapUserTerminated Property

Syntax	CapUserTerminated: <i>boolean</i> { read-only, access after open }
Remarks	If true, the user is able to terminate signature capture by checking a completion box, pressing a completion button, or performing some other interaction with the device. If false, the application must end signature capture by calling the endCapture method. 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 16.

DeviceEnabled Property (Common)

Syntax	DeviceEnabled: <i>boolean</i> { read-write, access after open-claim }
Remarks	If true, the signature capture device is enabled. If CapDisplay is true, then the display screen of the device is cleared. 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 16.

MaximumX Property

Syntax	MaximumX: <i>int32</i> { read-only, access after open }
Remarks	Holds the maximum horizontal coordinate of the signature capture device. It must be less than 65,536. 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 16.

MaximumY Property

Syntax	MaximumY: <i>int32</i> { read-only, access after open }
Remarks	Holds the maximum vertical coordinate of the signature capture device. It must be less than 65,536. 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 16.

PointArray Property**Updated in Release 1.7**

Syntax	PointArray: <i>array-of-points</i> { read-only, access after open-claim-enable } ¹
Remarks	<p>Holds the signature captured from the device. It consists of an array of (<i>x</i>, <i>y</i>) coordinate points. Each point is represented by four characters: <i>x</i> (low 8 bits), <i>x</i> (high 8 bits), <i>y</i> (low 8 bits), <i>y</i> (high 8 bits).</p> <p>A special point value is (0xFFFF, 0xFFFF) which indicates the end of a line (that is, a pen lift). Almost all signatures are comprised of more than one line.</p> <p>If RealTimeDataEnabled is false, then this property contains the entire captured signature. If RealTimeDataEnabled is true, then this property contains at least one point of the signature. The actual number of points delivered at one time is implementation dependent. The points from multiple data events are logically concatenated to form the entire signature, such that the last point from a data event is followed immediately by the first point of the next data event.</p> <p>The point representation definition is the same regardless of whether the signature is presented as a single PointArray, or as a series of real time PointArrays.</p> <p>Reconstruction of the signature using the points is accomplished by beginning a line from the first point in the signature to the second point, then to the third, and so on. When an end-of-line point is encountered, the drawing of the line ends, and the next line is drawn beginning with the next point. An end-of-line point is assumed (but need not be present in PointArray) at the end of the signature.</p> <p>This property is set prior to a DataEvent being delivered to the application or by the endCapture method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	RawData Property.

¹. In the OPOS environment, the format of this data depends upon the value of the **BinaryConversion** property. See **BinaryConversion** property on page A-28.

RawData Property**Updated in Release 1.7**

- Syntax** **RawData:** *binary* { **read-only, access after open-claim-enable** }²
- Remarks** Holds the signature captured from the device in a device-specific format.
- This data is often in a compressed form to minimize signature storage requirements. Reconstruction of the signature from this data requires device-specific processing.
- This property is set prior to a **DataEvent** being delivered to the application or by the **endCapture** method.
- Errors** A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
- See Also** **PointArray** Property.

RealTimeDataEnabled Property

- Syntax** **RealTimeDataEnabled:** *boolean* { **read-write, access after open** }
- Remarks** If true and **CapRealTimeData** is true, a series of partial signature data events is enqueued as the signature is captured until signature capture is terminated. Otherwise, the captured signature is enqueued as a single **DataEvent** when signature capture is terminated.
- Setting **RealTimeDataEnabled** will not cause any change in system behavior until a subsequent **beginCapture** method is performed. This prevents confusion regarding what would happen if it were modified between a **beginCapture** - **endCapture** pairing.
- 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 16.
- Some possible values of the exception’s *ErrorCode* property are:
- | Value | Meaning |
|--------------|---|
| E_ILLEGAL | Cannot set to true because CapRealTimeData is false. |
- See Also** **CapRealTimeData** Property, **beginCapture** Method, **endCapture** Method.

². In the OPOS environment, the format of this data depends upon the value of the **BinaryConversion** property. See **BinaryConversion** property on page A-28.

Methods (UML operations)

beginCapture Method

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

Parameter	Description
<i>formName</i>	The parameter contains the platform specific location for obtaining form or data screen information for display on the device screen.

Remarks Starts capturing a signature.

If **CapDisplay** is true, then *formName* is used to find information about the form or data screen to be displayed. The format and features of each signature capture device's form/data screen varies widely and is often built with proprietary tools. Therefore, this location's data, and possibly additional values and data, contain information that varies by Service. Typically, the contents of this data are set to a form/data screen file name, and extra values and data are set as needed to control its display.

After displaying the form or data screen, when applicable, the signature capture stylus is enabled.

Errors A UposException may be thrown when this method is invoked. For further information, see "Errors" on page 16.

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

Value	Meaning
E_NOEXIST	<i>formName</i> was not found.

See Also **CapDisplay** Property, **endCapture** Method.

endCapture Method

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

Remarks Stops (terminates) capturing a signature.

If **RealTimeDataEnabled** is false and a signature was captured, then it is placed in the properties **PointArray** and **RawData**. If no signature was captured, then **PointArray** and **RawData** are set to a length of zero.

If **RealTimeDataEnabled** is true and there are signature points remaining which have not been delivered to the application by a **DataEvent**, then the remaining signature is placed into the properties **PointArray** and **RawData**. If no signature was captured or all signature points have been delivered to the application, then **PointArray** and **RawData** are set to a length of zero.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	Signature capture was not in progress.

See Also **PointArray** Property, **RawData** Property, **RealTimeDataEnabled** Property, **beginCapture** Method, **DataEvent**.

Events (UML interfaces)

DataEvent

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

Description Notifies the application that input data is available.

Attributes This event contains the following attribute:

Attribute	Type	Description
<i>Status</i>	<i>int32</i>	Non-zero if the user has entered a signature before terminating capture. Zero if the user terminated capture with no signature.

Remarks This event can only be enqueued if the user can terminate signature capture – that is, if **CapUserTerminated** is true or **RealTimeDataEnabled** is true.
The properties **PointArray** and **RawData** are set to appropriate values prior to a **DataEvent** being delivered to the application.

See Also **endCapture** Method, “Events” on page 15.

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 Signature Capture Service to provide events to the application that are not otherwise supported by the Device 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 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 Signature Capture devices which may not have any knowledge of the Service’s need for this event.

See Also “Events” on page 15, **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 Signature Capture device error has been detected and a suitable response by the application 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> on page 16.
<i>ErrorCodeExtended</i>	<i>int32</i>	Extended Error Code causing the error event. This 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 has one of the following values:

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. (Very unlikely – see Remarks .)

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 **DataEventEnabled** is again set to true, then another **ErrorEvent** is enqueued with locus `EL_INPUT`.
 Default when locus is `EL_INPUT_DATA`.

- Remarks** Enqueued when an error is detected while trying to read signature capture data. This event is not delivered until **DataEventEnabled** is true and other event delivery requirements are met, so that proper application sequencing occurs.
- See Also** “Device Input Model” on page 18, “Device Information Reporting Model” on page 26, “Events” on page 15.

StatusUpdateEvent

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

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

Attributes This event contains the following attribute:

Attribute	Type	Description
<i>Status</i>	<i>int32</i>	Reports a change in the power state of a Signature Capture device.

Note that Release 1.3 added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See “StatusUpdateEvent” description on page 63.

- Remarks** Enqueued when the Signature Capture device detects a power state change.
- See Also** “Events” on page 15

Smart Card Reader / Writer

This Chapter defines the Smart Card Reader / Writer (SCR/W) device category.

Summary

Properties (UML attributes)

<i>Common</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
AutoDisable:	<i>boolean</i>	{ read-write }	1.8	Not Supported
CapPowerReporting:	<i>int32</i>	{ read-only }	1.8	open
CapStatisticsReporting:	<i>boolean</i>	{ read-only }	1.8	open
CapUpdateStatistics:	<i>boolean</i>	{ read-only }	1.8	open
CheckHealthText:	<i>string</i>	{ read-only }	1.8	open
Claimed:	<i>boolean</i>	{ read-only }	1.8	open
DataCount:	<i>int32</i>	{ read-only }	1.8	open
DataEventEnabled:	<i>boolean</i>	{ read-write }	1.8	open
DeviceEnabled:	<i>boolean</i>	{ read-write }	1.8	open & claim
FreezeEvents:	<i>boolean</i>	{ read-write }	1.8	open
OutputID:	<i>int32</i>	{ read-only }	1.8	open
PowerNotify:	<i>int32</i>	{ read-write }	1.8	open
PowerState:	<i>int32</i>	{ read-only }	1.8	open
State:	<i>int32</i>	{ read-only }	1.8	--
DeviceControlDescription:	<i>string</i>	{ read-only }	1.8	--
DeviceControlVersion:	<i>int32</i>	{ read-only }	1.8	--
DeviceServiceDescription:	<i>string</i>	{ read-only }	1.8	open
DeviceServiceVersion:	<i>int32</i>	{ read-only }	1.8	open
PhysicalDeviceDescription:	<i>string</i>	{ read-only }	1.8	open
PhysicalDeviceName:	<i>string</i>	{ read-only }	1.8	open

Properties (Continued)

<i>Specific:</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
CapCardErrorDetection:	<i>boolean</i>	{ read-only }	1.8	open
CapInterfaceMode:	<i>int32</i>	{ read-only }	1.8	open
CapIsoEmvMode:	<i>int32</i>	{ read-only }	1.8	open
CapSCPresentSensor:	<i>int32</i>	{ read-only }	1.8	open
CapSCSlots:	<i>int32</i>	{ read-only }	1.8	open
CapTransmissionProtocol:	<i>int32</i>	{ read-only }	1.8	open
InterfaceMode:	<i>int32</i>	{ read-write }	1.8	open, claim, & enable
IsoEmvMode:	<i>int32</i>	{ read-write }	1.8	open, claim, & enable
SCPresentSensor:	<i>int32</i>	{ read-only }	1.8	open, claim, & enable
SCSlot:	<i>int32</i>	{ read-write }	1.8	open, claim, & enable
TransactionInProgress:	<i>boolean</i>	{ read-only }	1.8	open
TransmissionProtocol:	<i>int32</i>	{ read-only }	1.8	open

Methods (UML operations)**Common**

<i>Name</i>	<i>Version</i>
open (logicalDeviceName: <i>string</i>): void { raises-exception }	1.8
close (): void { raises-exception, use after open }	1.8
claim (timeout: <i>int32</i>): void { raises-exception, use after open }	1.8
release (): void { raises-exception, use after open, claim }	1.8
checkHealth (level: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.8
clearInput (): void { raises-exception, use after open, claim }	1.8
clearOutput (): void { raises-exception, use after open, claim }	1.8
directIO (command: <i>int32</i>, inout data: <i>int32</i>, inout obj: <i>object</i>): void { raises-exception, use after open }	1.8
resetStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
retrieveStatistics (inout statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
updateStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8

Specific

<i>Name</i>	<i>Version</i>
beginInsertion (timeout: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.8
beginRemoval (timeout: <i>int32</i>): void { raises-exception, use after open, claim, enable }	1.8
endInsertion (): void { raises-exception, use after open, claim, enable }	1.8
endRemoval (): void { raises-exception, use after open, claim, enable }	1.8
readData (action: <i>int32</i>, inout count: <i>int32</i>, inout data: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
writeData (action: <i>int32</i>, count: <i>int32</i>, data: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>
upos::events::DataEvent			1.8
Status:	<i>int32</i>	{ read-only }	
upos::events::DirectIOEvent			1.8
EventNumber:	<i>int32</i>	{ read-only }	
Data:	<i>int32</i>	{ read-write }	
Obj:	<i>object</i>	{ read-write }	
upos::events::ErrorEvent			1.8
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			1.8
OutputID:	<i>int32</i>	{ read-only }	
upos::events::StatusUpdateEvent			1.8
Status:	<i>int32</i>	{ read-only }	

General Information

The Smart Card Reader / Writer programmatic name is “SmartCardRW”. This device was introduced in Version 1.8 of the specification.

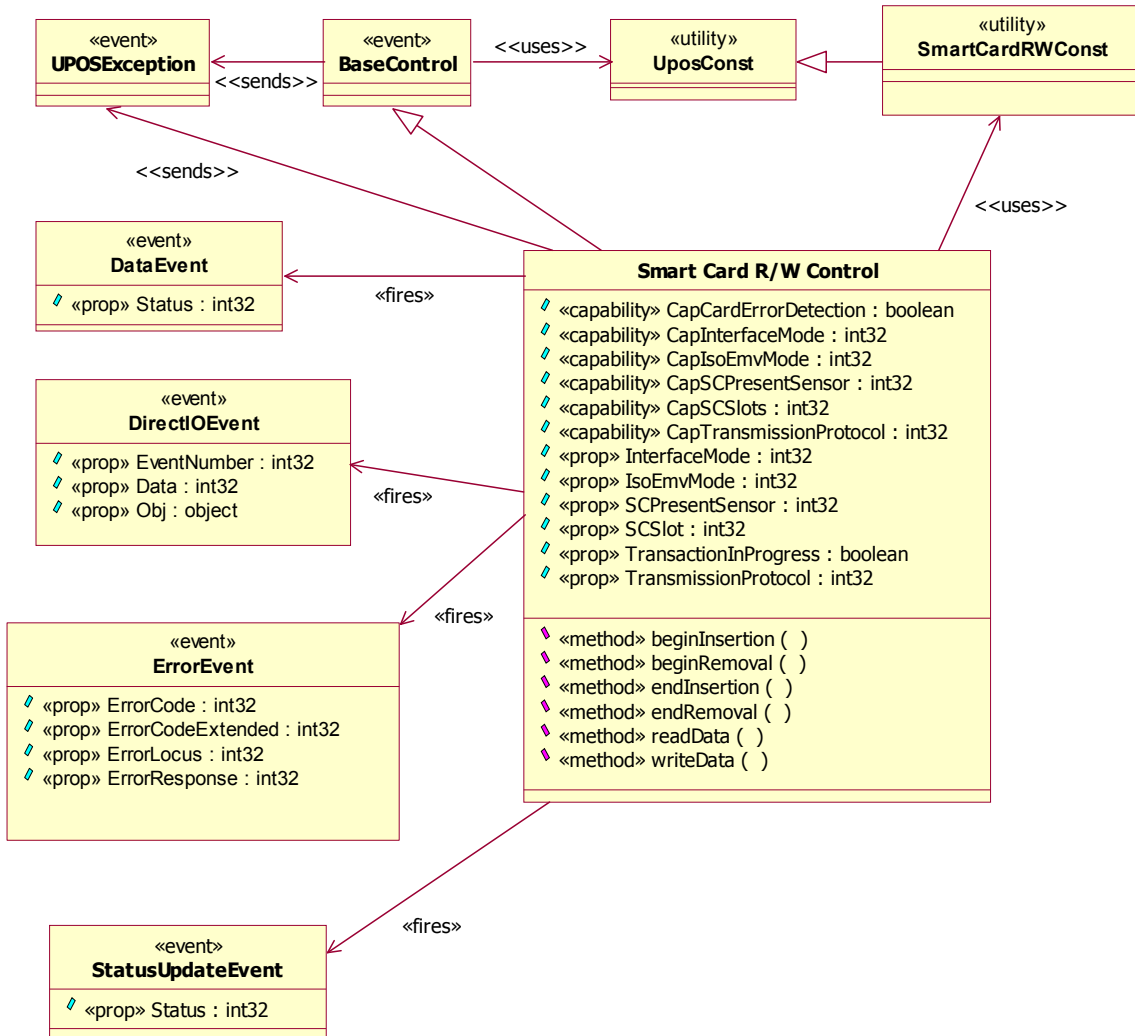
Capabilities

The Smart Card Reader / Writer (SCR/W) device has the following capabilities.

- Support for the reading and writing of Smart Cards that conform to the ISO/IEC 7816 standard (contact type) and ISO/IEC 14443 (contactless type).
- Interface with simple memory cards, protected or segmented memory cards, stored value memory cards, and CPU/MPU multifunction cards.
- Functions are limited to the actual Smart Card read and write operations only. Full function type devices such as a “Payment Terminal” (defined as a unit that incorporates a SCR/W plus additional devices such as a Pin Pad, Display, Signature Capture, and MSR reader in an integrated device) are not covered in this peripheral class.
- Support for Smart Cards that use physical electrical contacts and/or close range Radio Frequency to exchange power and data.
- Ability to sense when a card is present or absent is supported.
- Optional support of Security Application Modules (SAM) for CPU/MPU cards may be provided.
- Up to four types of API communication methods to the SCR/W may be supported:
 1. **Command and Data Mode:** Very basic ASCII format for commands and data interchange.
 2. **Data Block Mode:** A block of string data that contains commands and data is sent to the SCR/W Device Service. The application and the SCR/W Service need to agree upon a communication protocol and data format before using this mode.
 3. **APDU Mode:** Same as Data Block Mode except that the block of string data that contains commands and data sent to the SCR/W Service conforms to the ISO/IEC 7816 APDU (Application Protocol Data Units) standard for smart cards. ISO and EMV messaging formats are supported and selectable if the SCR/W has the capability to switch to one of these formats.
 4. **XML Data Block Mode:** A block of string data that contains commands and data is sent to the SCR/W Service. The application and the SCR/W Service agree to use a communication protocol and data format defined in this standard consistent with the XML Data Dictionary and XML schema guidelines as outlined in the NRF-ARTS IXRetail XML standard.

Smart Card Reader / Writer Class Diagram

The following diagram shows the relationships between the SCR/W classes.



Model

The general model of Smart Card Reader / Writer is as follows:

- The Smart Card Reader / Writer (SCR/W) device has a wide range of usages that depend upon a variety of ISO 7816 compliant smart cards. These include cards with or without physical electrical contacts and proximity types that may function as memory cards, processor cards (T0 and/or T1 **TransmissionProtocol**), electronic purse cards, security access module (SAM) processor cards, and security cards. The SCR/W scope is limited to providing access to the smart card so that data retrieval, data storage, or program execution on the smart card can be implemented.
- It is the responsibility of the application to have knowledge of what type of Smart Card transactions the SCR/W device will allow. To help facilitate a wide range of possibilities of usage, four different communication command and data interchange methods (**InterfaceMode**) are provided. As part of the initialization sequence, the application should query the **CapInterfaceMode** to determine what is allowed and set the **InterfaceMode** property to the mode that will be used.
- To begin operation, the application must call the **open** and **claim** methods to set up a communication path to the SCR/W device. When the application is ready to interact with a smart card, the **DeviceEnabled** property must be set to true. Then the SCR/W is able to accept a smart card; a **StatusUpdateEvent** is fired when one has been detected.

The **beginInsertion** method, with its time-out value set to some finite value, provides a way to allow the application to wait for a smart card to be detected. If the time-out value expires, the program must call another **beginInsertion** method to continue its quest for detecting a smart card. Once the smart card has been detected, the application must call the **endInsertion** method.

- **Input**

If the application requests data from the smart card, the **readData** method must be called. When data is available, a **DataEvent** is enqueued. The application must set the **DataEventEnabled** property to true in order for the **DataEvent** to be delivered.

If an error occurs while reading the smart card's data, 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**

The **writeData** method is always performed asynchronously. All output data is performed on a first-in, first-out basis. When the application calls the **writeData** method, the SCR/W buffers the request and begins the communication process through the SCR/W device to the smart card.

Depending upon the **InterfaceMode** property, the **writeData** method data is either parsed by the Service or passed natively directly to the SCR/W device and then on to the smart card. A unique identification number is assigned for the data associated with the **writeData** call and is stored in the **OutputID** property. The data is enqueued for delivery to the SCR/W device as soon as it can receive and process it.

When the **writeData** method completes sending the data associated with the current output request, an **OutputCompleteEvent** is delivered to the application. The **OutputID** associated with this output request is contained in the **OutputCompleteEvent**.

If the **writeData** method fails during data transfer, an **ErrorEvent** will be delivered to the application. If the application had multiple outstanding output requests, the **OutputID** of the failed request is determined by evaluating the *OutputID* associated with the last successful **OutputCompleteEvent**. The request that failed is the one that was issued immediately after the last request that successfully completed.

All buffered output data may be deleted by calling the **clearOutput** method. This 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 smart card, the application must call the **beginRemoval** method, specifying a timeout value. If the card is not removed before the timeout period elapses, the SCR/W fires an exception. The application must call the **beginRemoval** method again until the smart card is removed from the SCR/W device.

When the smart card is no longer detected in the SCR/W, a **StatusUpdateEvent** is fired.

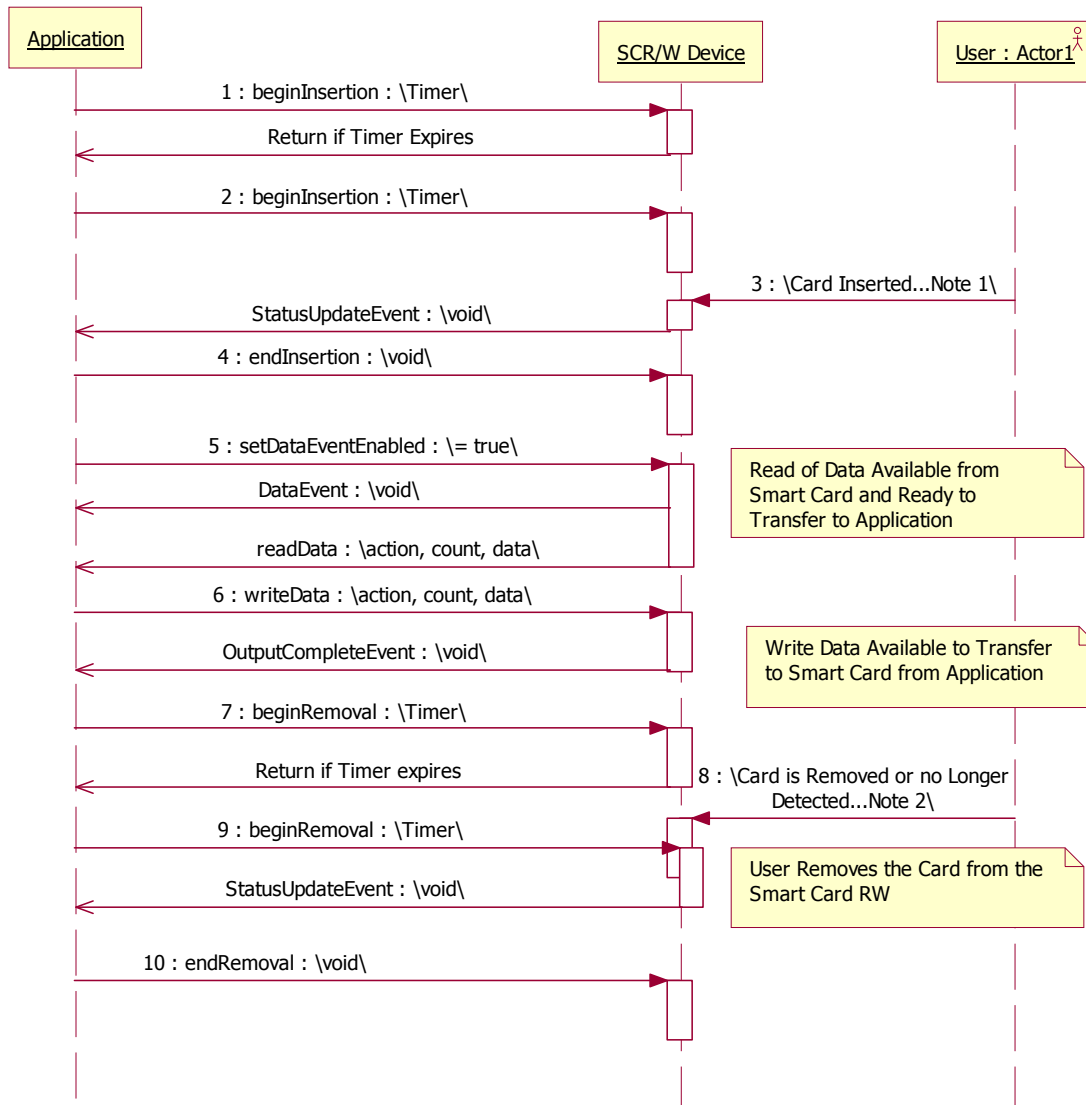
To exit the removal mode, either after the card was physically removed or the application aborts the smart card removal process, the application must call the **endRemoval** method.

When the application is finished using the SCR/W device, the application must set the **DeviceEnabled** property to false and call the **release** method. If no further interaction with the SCR/W device is required, the application must call the **close** method.

There may be times when the smart card is extracted from the SCR/W device before the normal usage sequence has been completed. This is referred to as having the card “torn” from the SCR/W device. The application will receive a **StatusUpdateEvent** indicating the card is no longer “present”. In addition the **SCPpresentSensor** property would have been set to false.

Card Insertion Diagram

The processing from card insertion to card removal is shown below. All methods, other than **writeData**, are performed synchronously.



- (1) If the smart card is not inserted into the SCR/W before the application specified timeout elapses, an exception is fired. The application needs to call **beginInsertion** again to confirm that a smart card has been inserted or call **endInsertion** to cancel the card insertion. After a successful **beginInsertion**, the application must call **endInsertion** to cause the SCR/W to exit insertion mode and allow for further **readData**, **writeData**, or other methods to be used with the SCR/W to obtain data from the smart card. When a card is detected, a **StatusUpdateEvent** is fired.

- (2) If the smart card is not removed from the SCR/W before the application specified timeout elapses, an exception is fired. The application needs to call **beginRemoval** again to confirm that the smart card has been removed, or call **endRemoval** to cancel the card removal. After a successful **beginRemoval**, the application must call **endRemoval** to cause the SCR/W to exit removal mode. When a card is no longer detected, a **StatusUpdateEvent** is fired.

Device Sharing

The SCR/W 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 of the SCR/W 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 Transfer Modes

The SCR/W has the flexibility to be able to operate in one or more modes to enable the transfer of data to and from the smart card. When the SCR/W is initialized, the application must determine what communication and operation mode will be used based upon a query of the capabilities of the SCR/W device. The **InterfaceMode** property is used to store the current communication mode.

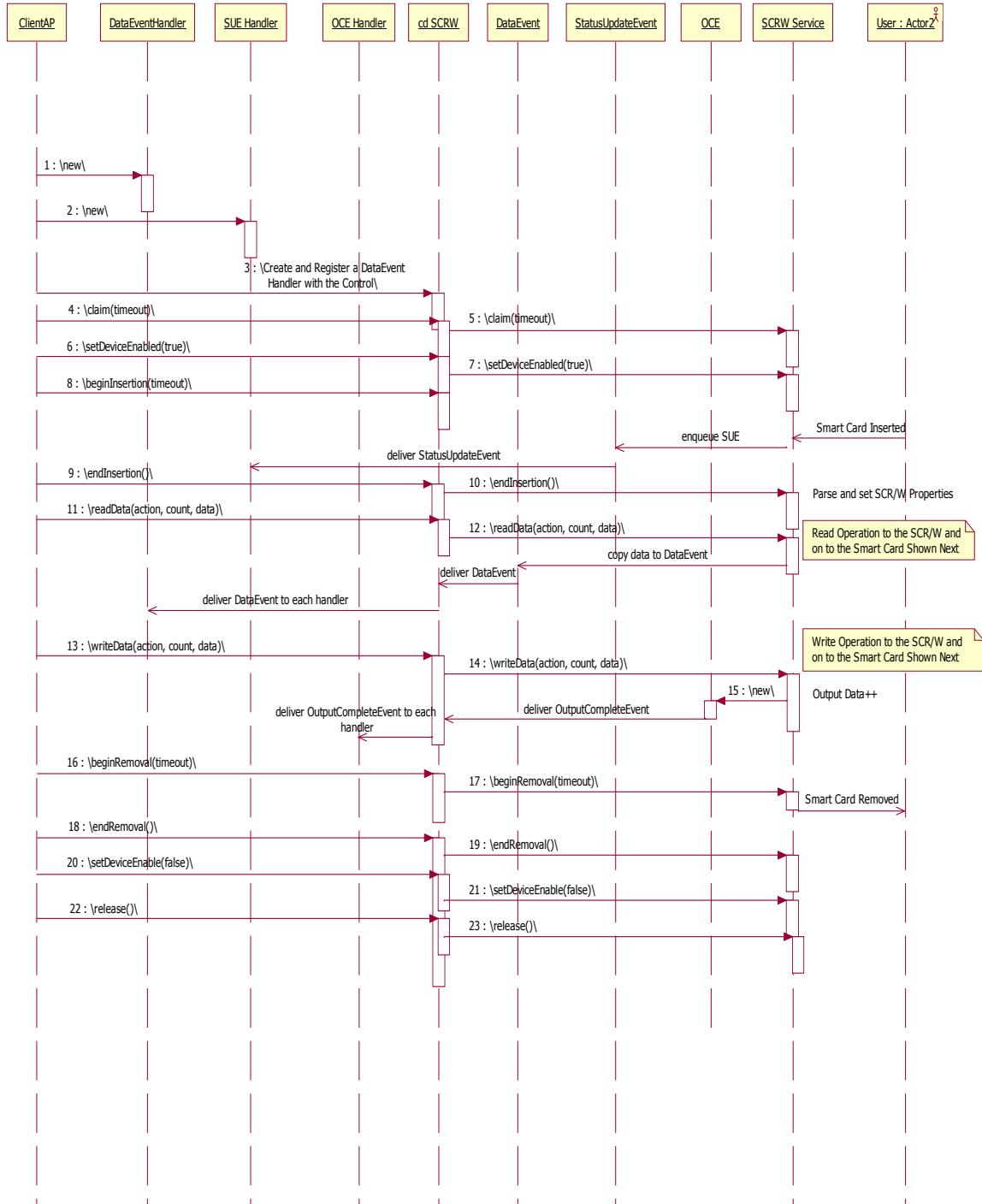
In the *Command / Data* mode, a simple read and write data functionality is defined between the application and the SCR/W. The commands will cause the data to be retrieved from, placed onto, or placed onto and executed on the smart card currently available to the SCR/W device. Greater knowledge of the specific SCR/W device is required in this mode. The application should query the **PhysicalDeviceName** and/or **PhysicalDeviceDescription** properties and create the write data and resultant read data based upon the type of SCR/W that is connected to the system.

In the *Block Transfer* mode, blocks of commands/data are sent to and retrieved from the SCR/W Service. It is up to the Service to parse the commands and data from the block of information sent to it from the application and invoke the necessary function and response in the smart card currently in the SCR/W. Knowledge of the message content between the application and the SCR/W must be established when the open method is called. The application should query the **PhysicalDeviceName** and/or **PhysicalDeviceDescription** properties and base its message content upon the type of SCR/W that is connected to the system.

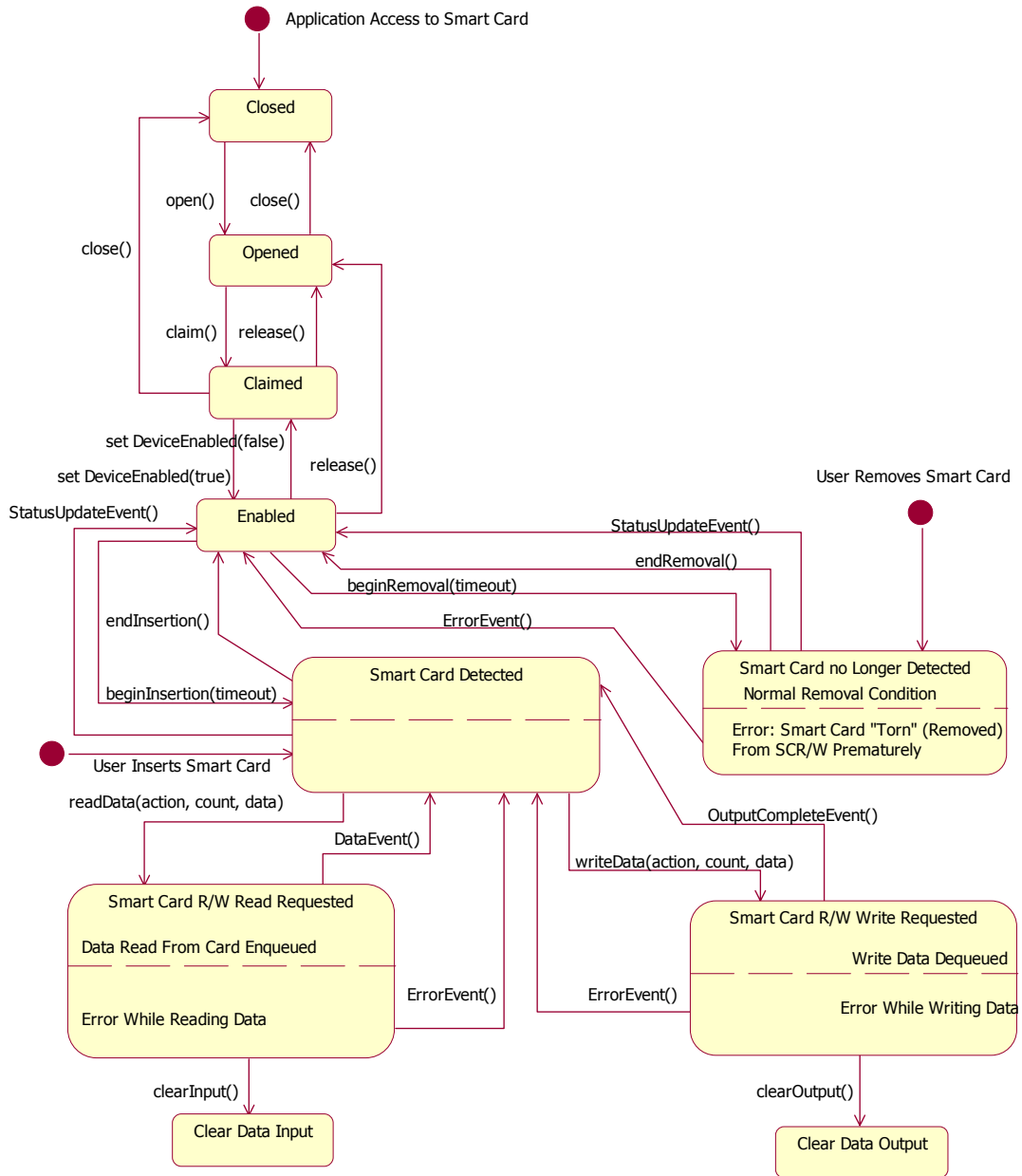
In the *APDU Transfer* mode, blocks of data are sent to and retrieved from the SCR/W Service similar to the Block Transfer mode described above. However, in this mode the commands and data consist of string data elements that comply to the ISO/IEC 7816 APDU (Application Protocol Data Units) standard for Smart Cards communication. Provision has been made to support the messaging requirements of ISO or EMV for operating in the APDU mode. The **CapIsoEmvMode** property can be queried to determine what modes are supported by the device. The application then sets the **IsoEmvMode** property to the desired messaging scheme prior to sending data to and receiving data from the SCR/W device.

In the *XML Block Transfer* mode, blocks of data are sent to and retrieved from the SCR/W Service similar to the Block Transfer mode described above. However, in this mode the commands and data are in the form of XML messages. The data elements and schemas of these messages conform to the IXRetail Standard XML messaging as they apply to the SCR/W device.

Smart Card Reader / Writer Sequence Diagram



Smart Card Reader / Writer State Diagram



Properties (UML Attributes)

CapCardErrorDetection Property

Syntax	CapCardErrorDetection: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the SCR/W has the ability to report that the smart card has been “torn” (removed before all transfers have been completed) from the device, false if it does not. The ErrorEvent is only fired with the <i>ErrorCode</i> set to the value “ESC_TORN” if a “torn” error is detected and the value for this property is true. 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 16.
See Also	ErrorEvent event.

CapInterfaceMode Property

Syntax	CapInterfaceMode: <i>int32</i> { read-only, access after open }										
Remarks	This capability indicates the types of interface modes that the SCR/W device is capable of supporting, a simple transaction command and data mode, a block data mode, APDU format block data mode, or a block XML data mode that uses the IXRetail Standard for SCR/W functionality. The InterfaceMode property will reflect the currently selected interface mode that the application is using to communicate with the device. CapInterfaceMode is a bitwise logical OR combination of any of the following values:										
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>SC_CMODE_TRANS</td> <td>Simple Transaction Command and Data Mode</td> </tr> <tr> <td>SC_CMODE_BLOCK</td> <td>Block Data Mode</td> </tr> <tr> <td>SC_CMODE_APDU</td> <td>Same as Block Data Mode except APDU Standard Commands are used.</td> </tr> <tr> <td>SC_CMODE_XML</td> <td>XML Block Data Mode</td> </tr> </tbody> </table> <p>This property is initialized by the open method.</p>	Value	Meaning	SC_CMODE_TRANS	Simple Transaction Command and Data Mode	SC_CMODE_BLOCK	Block Data Mode	SC_CMODE_APDU	Same as Block Data Mode except APDU Standard Commands are used.	SC_CMODE_XML	XML Block Data Mode
Value	Meaning										
SC_CMODE_TRANS	Simple Transaction Command and Data Mode										
SC_CMODE_BLOCK	Block Data Mode										
SC_CMODE_APDU	Same as Block Data Mode except APDU Standard Commands are used.										
SC_CMODE_XML	XML Block Data Mode										
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.										
See Also	InterfaceMode Property, IsoEmvMode Property.										

CapIsoEmvMode Property

- Syntax** **CapIsoEmvMode:** *int32* { read-only, access after open }
- Remarks** This capability indicates the message modes the SCR/W supports in order to interoperate with a smart card when the **InterfaceMode** is set to SC_MODE_APDU. The APDU messaging format is dependent upon whether the ISO or EMV standard is desired to be used. The **IsoEmvMode** property is used to select the APDU mode that the SCR/W is currently using to interoperate with the smart card.
- CapIsoEmvMode** is a bitwise logical OR combination of any of the following values:
- | Value | Meaning |
|--------------|---|
| SC_CMODE_ISO | APDU messaging format conforms to the ISO standard. |
| SC_CMODE_EMV | APDU messaging format conforms to the EMV standard. |
- 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 16.
- See Also** **IsoEmvMode** Property, **InterfaceMode** Property.

CapSCPresentSensor Property

- Syntax** **CapSCPresentSensor:** *int32* { read-only, access after open }
- Remarks** This capability indicates if the SCR/W device can sense if a smart card is present in one of the available slots (entry points and/or proximity zones) where a user can insert a smart card.
- The SCR/W device will always have a minimum of one slot available (designated as the default slot) indicated by the LSB.
- CapSCPresentSensor** is a bitwise logical OR combination of any of the *int32* bits with bit 0 (LSB) slot 0 (default); bit 1, slot 1; bit 2, slot 2; etc. If the bit value is one, then the SCR/W has a sensor that can detect when a smart card is present; the bit value is zero 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 16.
- See Also** **SCPresentSensor** Property.

CapSCSlots Property

- Syntax** **CapSCSlots: *int32* { read-only, access after open }**
- Remarks** This capability indicates the number of available slots (entry points and/or proximity zones) where a user can insert a smart card for detection in the SCR/W device. The application can select the slot to use by setting the **SCSlot** property to one of the allowable **CapSCSlots** values. The device will always have a minimum of one slot available (designated as the default slot) indicated by the LSB set to one.
- CapSCSlots** is a bitwise logical OR combination of any of the *int32* bits with bit 0 (LSB) slot 0 (default); bit 1, slot 1; bit 2, slot 2; etc.
- 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 16.
- See Also** **SCSlot** Property.

CapTransmissionProtocol Property

- Syntax** **CapTransmissionProtocol: *int32* { read-only, access after open }**
- Remarks** This capability indicates the types of ISO 7816-3 transmission protocols that the SCR/W device is capable of supporting, T=0 (asynchronous half duplex character transmission protocol), T=1 (asynchronous half duplex block transmission protocol). The **TransmissionProtocol** property will reflect the currently selected transmission protocol being used to communicate with the device.
- CapTransmissionProtocol** is a bitwise logical OR combination of any of the following values:
- | Value | Meaning |
|-----------------------|--|
| SC_CTRANS_PROTOCOL_T0 | Asynchronous, Half Duplex, Character, Transmission Protocol Mode |
| SC_CTRANS_PROTOCOL_T1 | Asynchronous, Half Duplex, Block Transmission Protocol 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 16.
- See Also** **TransmissionProtocol** Property.

InterfaceMode Property

- Syntax** **InterfaceMode: *int32* { read-write, access after open-claim-enable }**
- Remarks** This property indicates the current communication interface mode that the SCR/W device is using to communicate with the application program. The property **CapInterfaceMode** contains the interface modes that are supported by the SCR/W Service. If an **InterfaceMode** is selected that is not consistent with **CapInterfaceMode**, a `UposException` will be thrown.

InterfaceMode may be one of the following values:

Value	Meaning
SC_MODE_TRANS	Simple Transaction Command and Data Mode
SC_MODE_BLOCK	Block Data Mode
SC_MODE_APDU	Same as Block Data Mode except APDU Standard Defines the Commands and data.
SC_MODE_XML	XML Block Data Mode

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

- Errors** A `UposException` may be thrown when this property is accessed. For further information, see “Errors” on page 16.
- See Also** **CapInterfaceMode** Property.

IsoEmvMode Property

- Syntax** **IsoEmvMode: *int32* { read-only, access after open-claim-enable }**
- Remarks** This property indicates the message modes the SCR/W is currently using in order to interoperate with a smart card when the **InterfaceMode** is set to SC_MODE_APDU. The APDU messaging format is dependent upon whether the ISO or EMV standard is desired to be used. The **CapIsoEmvMode** capability defines the available modes the SCR/W supports and the **IsoEmvMode** property will be set to reflect the mode that is currently in use by the SCR/W device.

IsoEmvMode may be one of the following values:

Value	Meaning
SC_MODE_ISO	APDU messaging format currently in use conforms to the ISO standard.
SC_MODE_EMV	APDU messaging format currently in use conforms to the EMV standard.

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 16.
- See Also** **CapIsoEmvMode** Property, **InterfaceMode** Property.

SCPresentSensor Property

Syntax	SCPresentSensor: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	<p>This property indicates that a smart card has been detected in one of the supported slots present in the SCR/W device and is in a position to exchange data with the application. This property is only active if the CapSCPresentSensor confirms that a smart card present sensor is supported by the slot. The SCR/W device will always have a minimum of one slot available (designated as the default slot) indicated by the LSB but may or may not support a smart card present sensor.</p> <p>SCPresentSensor is a bitwise logical OR combination of any of the <i>int32</i> bits with bit 0 (LSB) slot 0 (default); bit 1, slot 1; bit 2, slot 2; etc. If the bit value is one, then the sensor indicates that a smart card is present; the bit value is zero 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 16.
See Also	CapSCPresentSensor Property.

SCSlot Property

Syntax	SCSlot: <i>int32</i> { read-write, access after open-claim-enable }
Remarks	<p>This property indicates the current slot (entry point or proximity zone) where a user can insert a smart card for detection in the SCR/W device. The application can select the slot to use by setting the SCSlot property to one of the allowable CapSCSlots values. The device will always have a minimum of one slot available (designated as the default, slot 0) indicated by the LSB set to one.</p> <p>SCSlot may be set by the application to one of the CapSCSlots values as follows:</p> <p>bit 0 (LSB) slot 0 (default); bit 1, slot 1; bit 2, slot 2; etc.</p> <p>This property is initialized by the open method to the default, slot 0 value.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.
See Also	CapSCSlots Property.

TransactionInProgress Property

Syntax	TransactionInProgress: <i>boolean</i> { read-only, access after open }
Remarks	If true, then a smart card has been detected and active interchange of information with the smart card is taking place. 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 16.
See Also	SCPresentSensor Property.

TransmissionProtocol Property

Syntax	TransmissionProtocol: <i>int32</i> { read-only, access after open }						
Remarks	This property indicates the type of ISO 7816-3 transmission protocols that the SCR/W device is currently supporting, T=0 (asynchronous half duplex character transmission protocol) or T=1 (asynchronous half duplex block transmission protocol). The TransmissionProtocol property will reflect the currently selected transmission protocol being used to communicate with the device. TransmissionProtocol is a bitwise data element based upon the supported modes as defined by the CapTransmissionProtocol property and may be one of the following values:						
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>SC_TRANS_PROTOCOL_T0</td> <td>Asynchronous, Half Duplex, Character, Transmission Protocol Mode</td> </tr> <tr> <td>SC_TRANS_PROTOCOL_T1</td> <td>Asynchronous, Half Duplex, Block Transmission Protocol Mode</td> </tr> </tbody> </table>	Value	Meaning	SC_TRANS_PROTOCOL_T0	Asynchronous, Half Duplex, Character, Transmission Protocol Mode	SC_TRANS_PROTOCOL_T1	Asynchronous, Half Duplex, Block Transmission Protocol Mode
Value	Meaning						
SC_TRANS_PROTOCOL_T0	Asynchronous, Half Duplex, Character, Transmission Protocol Mode						
SC_TRANS_PROTOCOL_T1	Asynchronous, Half Duplex, Block Transmission Protocol 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 16.						
See Also	CapTransmissionProtocol 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 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 smart card is inserted or an error occurs.

Remarks Called to initiate smart card insertion processing in either a contact type or contactless type SCR/W.

When called, SCR/W state is changed to allow the insertion of a smart card and the smart card insertion mode is entered. This method is paired with the **endInsertion** method for controlling smart card insertion.

If the SCR/W device cannot be placed into insertion mode an exception is raised. Otherwise, the Control continues to monitor smart card insertion until either the smart card is not inserted before *timeout* milliseconds have elapsed, or an error is reported by the SCR/W device. In the latter case, the Control raises an exception with the appropriate error code. The SCR/W device remains in smart card insertion mode. This allows an application to perform some user interaction and reissue the **beginInsertion** method without altering the smart card handling mechanism.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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 SCR/W does not exist or an invalid <i>timeout</i> parameter was specified.
E_TIMEOUT	The specified time has elapsed without the smart card being properly inserted.
E_EXTENDED	Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 849.

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 smart card is removed or an error occurs.

Remarks Called to initiate smart card removal processing.

When called, the SCR/W is made ready to be removed from either a contact type or a contactless type SCR/W. This method is paired with the **endRemoval** method for controlling smart card removal.

The contact type model that has the sensor in the entrance ends normally when a card is removed from SCR/W. The contactless model (without a sensor) ends normally when the smart card has been removed from the proximity of the SCR/W device.

If the SCR/W cannot be placed into removal or ejection mode, an exception is raised. Otherwise, the Control continues to monitor smart card removal until either the smart card is not ejected before *timeout* milliseconds have elapsed, or an error is reported by the SCR/W. In this case, the Control raises an exception with the appropriate error code. The SCR/W remains in smart card ejection mode. This allows an application to perform some user interaction and reissue the **beginRemoval** method without altering the smart card handling mechanism.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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 SCR/W does not exist or an invalid <i>timeout</i> parameter was specified.
E_TIMEOUT	The specified time has elapsed without the smart card being properly removed.
E_EXTENDED	Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 849.

See Also **beginInsertion** Method, **endInsertion** Method, **endRemoval** Method.

endInsertion Method

Syntax	endInsertion (): void { raises-exception, use after open-claim-enable }								
Remarks	Called to end smart card insertion processing. When called, the SCR/W is taken out of smart card insertion mode. If no smart card is present, an exception is raised. This method is paired with the beginInsertion method for controlling smart card insertion in either a contact type or contactless type SCR/W.								
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16. Some possible values of the exception’s <i>ErrorCode</i> property are:								
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>E_ILLEGAL</td> <td>The SCR/W is not in smart card insertion mode.</td> </tr> <tr> <td>E_FAILURE</td> <td>A card is not inserted in the SCR/W.</td> </tr> <tr> <td>E_EXTENDED</td> <td>Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 849.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	The SCR/W is not in smart card insertion mode.	E_FAILURE	A card is not inserted in the SCR/W.	E_EXTENDED	Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 849.
Value	Meaning								
E_ILLEGAL	The SCR/W is not in smart card insertion mode.								
E_FAILURE	A card is not inserted in the SCR/W.								
E_EXTENDED	Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 849.								
See Also	beginInsertion Method, beginRemoval Method, endRemoval Method.								

endRemoval Method

Syntax	endRemoval (): void { raises-exception, use after open-claim-enable }								
Remarks	Called to end smart card removal processing. When called, the SCR/W is taken out of smart card removal mode in either a contact type or contactless type SCR/W. If a smart card is present, an exception is raised. This method is paired with the beginRemoval method for controlling smart card removal. The application may choose to call this method immediately after a successful beginRemoval if it wants to use the SCR/W sensors to determine when the smart card has been removed. Alternatively, the application may prompt the user and wait for a key being pressed before calling this method.								
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16. Some possible values of the exception’s <i>ErrorCode</i> property are:								
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>E_ILLEGAL</td> <td>The SCR/W is not in smart card removal mode.</td> </tr> <tr> <td>E_FAILURE</td> <td>There is a card in the SCR/W.</td> </tr> <tr> <td>E_EXTENDED</td> <td>Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 849.</td> </tr> </tbody> </table>	Value	Meaning	E_ILLEGAL	The SCR/W is not in smart card removal mode.	E_FAILURE	There is a card in the SCR/W.	E_EXTENDED	Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 849.
Value	Meaning								
E_ILLEGAL	The SCR/W is not in smart card removal mode.								
E_FAILURE	There is a card in the SCR/W.								
E_EXTENDED	Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 849.								
See Also	beginInsertion Method, beginRemoval Method, endInsertion Method.								

readData Method

Syntax **readData (action: *int32*, inout count: *int32*, inout data: *string*):**
 void { raises-exception, use after open-enable }

Parameter	Description
<i>action</i>	Indicates the type of processing of the data that is to be done by the smart card.
<i>count</i>	The total number of data bytes that are being returned by the smart card.
<i>data</i>	The data that is returned from the smart card.

Remarks Reads data from a smart card using the SCR/W.

The *action* parameter may have one of the following values:

Value	Meaning
SC_READ_DATA	The <i>data</i> being read from the smart card present in the SCR/W is from the Data Area on the smart card.
SC_READ_PROGRAM	The <i>data</i> being read from the smart card present in the SCR/W is an executable program that was found in the smart card memory associated with executable programs.
SC_EXECUTE_AND_READ_DATA	The <i>data</i> being read from the smart card present in the SCR/W is data that was processed by a program currently resident on the smart card. When this <i>action</i> is requested the smart card program will be started and send back the <i>data</i> that it has processed.
SC_XML_READ_BLOCK_DATA	The <i>data</i> being read is XML data that the SCR/W is sending to the application. It is up to the application to parse the data being returned.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_CLAIMED	Cannot read because the smart card present in the SCR/W is claimed by another application.
E_ILLEGAL	The <i>action</i> is not valid for the type of smart card present in the SCR/W or the <i>count</i> value is not valid for the smart card present in the SCR/W.

See Also **writeData** Method.

writeData Method

Syntax **writeData (action: *int32*, count: *int32*, data: *string*):
void { raises-exception, use after open-enable }**

Parameter	Description
<i>action</i>	Indicates the type of processing of the data that is to be done by the smart card.
<i>count</i>	The total number of data bytes that are being sent to the smart card with this method.
<i>data</i>	The data that is to be sent to the smart card.

Remarks Writes *data* to a smart card using the SCR/W.

The *action* parameter may have one of the following values:

Value	Meaning
SC_STORE_DATA	The <i>data</i> being sent to the smart card present in the SCR/W is to be stored in the Data Area on the smart card.
SC_STORE_PROGRAM	The <i>data</i> being sent to the smart card present in the SCR/W is an executable program and will be placed in the smart card memory associated with executable programs.
SC_EXECUTE_DATA	The <i>data</i> being sent to the smart card present in the SCR/W is data that will be processed by a program that is currently resident and can execute on the smart card. When this <i>action</i> is requested the smart card program will be started and will use the <i>data</i> that has been sent.
SC_XML_BLOCK_DATA	The <i>data</i> being sent is XML data and is to be parsed by the SCR/W to determine what actions are to take place.
SC_SECURITY_FUSE	The smart card present in the SCR/W will have its security fuse activated to prevent future data from being stored in the smart card.
SC_RESET	The smart card present in the SCR/W will be instructed to be reset to its “power on” state and ready to execute an application command.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_CLAIMED	Cannot write because the smart card present in the SCR/W is claimed by another application.
E_ILLEGAL	The <i>action</i> is not valid for the type of smart card present in the SCR/W or the <i>count</i> value is not valid for the smart card present in the SCR/W.

See Also **readData** Method.

Events (UML Interfaces)

DataEvent

<< event >> **upos::events::DataEvent**

Status: *int32* { read-only }

Description Fired to present input data from the SCR/W 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 smart card data is placed in each applicable 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 SCR/W 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 smart card devices which may not have any knowledge of the Service's need for this event.

See Also "Events" on page 15, **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 SCR/W error has been detected and a suitable response by the application is necessary to process the error condition.

Attributes This event contains the following attributes:

Attributes	Type	Description
<i>ErrorCode</i>	<i>int32</i>	Error code causing the error event. See a list of Error Codes on page 16.
<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
ESC_READ	There was a read error.
ESC_WRITE	There was a write error.
ESC_TORN	The smart card was prematurely removed from the SCR/W unexpectedly. <i>Note: CapCardErrorDetection</i> capability must be true before this can be set.
ESC_NO_CARD	There is no card detected in the SCR/W but a card was expected.

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 all buffered output data (including all asynchronous output) or buffered input data. The error state is exited. Default when locus is EL_INPUT.
ER_CONTINUE_INPUT	Used 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 data from a newly inserted smart 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 applicable properties before this event is delivered.

Output error events are generated and delivered when an error occurs during asynchronous **writeData** processing. The errors are placed into the applicable properties before the event is delivered.

See Also **CapCardErrorDetection** Property, **SCPPresentSensor** Property, **readData** method, **writeData** method.

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 21.

StatusUpdateEvent

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

Description Notifies the application that there is a change in the status of the SCR/W 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 SCR/W device.

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

Value	Meaning
SC_SUE_NO_CARD	No card detected in the SCR/W Device.
SC_SUE_CARD_PRESENT	There is a card in the device.

Remarks Fired when the status of a smart card in the SCR/W changes.

See Also "Events" on page 15.

Tone Indicator

This Chapter defines the Tone Indicator device category.

Summary

Properties (UML attributes)

<i>Common</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</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
CapStatisticsReporting:	<i>boolean</i>	{ read-only }	1.8	open
CapUpdateStatistics:	<i>boolean</i>	{ read-only }	1.8	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 }	1.2	Not Supported
DataEventEnabled:	<i>boolean</i>	{ read-write }	1.2	Not Supported
DeviceEnabled:	<i>boolean</i>	{ read-write }	1.2	open
FreezeEvents:	<i>boolean</i>	{ read-write }	1.2	open
OutputID:	<i>int32</i>	{ read-only }	1.2	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 }	1.2	--
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

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 }	1.2	open & enable
CapPitch:	<i>boolean</i>	{ read-only }	1.2	open
CapVolume:	<i>boolean</i>	{ read-only }	1.2	open
InterToneWait:	<i>int32</i>	{ read-write }	1.2	open & enable
Tone1Duration:	<i>int32</i>	{ read-write }	1.2	open & enable
Tone1Pitch:	<i>int32</i>	{ read-write }	1.2	open & enable
Tone1Volume:	<i>int32</i>	{ read-write }	1.2	open & enable
Tone2Duration:	<i>int32</i>	{ read-write }	1.2	open & enable
Tone2Pitch:	<i>int32</i>	{ read-write }	1.2	open & enable
Tone2Volume:	<i>int32</i>	{ read-write }	1.2	open & enable

Methods (UML operations)**Common**

<i>Name</i>	<i>Version</i>
open (logicalDeviceName: <i>string</i>): void { raises-exception }	1.2
close (): void { raises-exception, use after open }	1.2
claim (timeout: <i>int32</i>): void { raises-exception, use after open }	1.2
release (): void { raises-exception, use after open, claim }	1.2
checkHealth (level: <i>int32</i>): void { raises-exception, use after open, enable }	1.2
clearInput (): void { }	<i>Note</i> <i>Not supported</i>
clearOutput (): void { raises-exception, use after open, enable }	1.2
directIO (command: <i>int32</i>, inout data: <i>int32</i>, inout obj: <i>object</i>): void { raises-exception, use after open }	1.2
resetStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
retrieveStatistics (inout statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8
updateStatistics (statisticsBuffer: <i>string</i>): void { raises-exception, use after open, claim, enable }	1.8

Specific***Name***

sound (numberOfCycles: <i>int32</i>, interSoundWait: <i>int32</i>): void { raises-exception, use after open, enable }	<i>Note</i>	1.2
soundImmediate (): void { raises-exception, use after open, enable }	<i>Note</i>	1.2

Note: Also requires that no other application has claimed the tone indicator.

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>
upos::events::DataEvent		<i>Not Supported</i>	
upos::events::DirectIOEvent			1.2
EventNumber:	<i>int32</i>	{ read-only }	
Data:	<i>int32</i>	{ read-write }	
Obj:	<i>object</i>	{ read-write }	
upos::events::ErrorEvent			1.2
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			1.2
OutputID:	<i>int32</i>	{ read-only }	
upos::events::StatusUpdateEvent			1.3
Status:	<i>int32</i>	{ read-only }	

General Information

The Tone Indicator programmatic name is “ToneIndicator”.

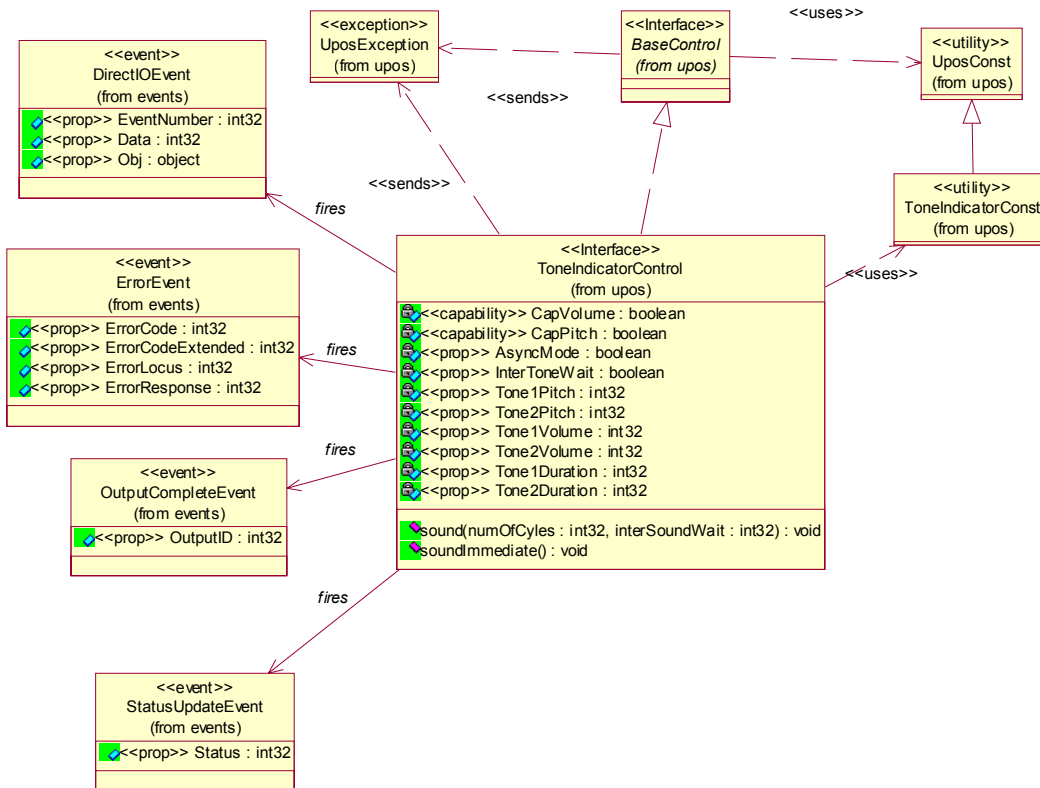
Capabilities

The Tone Indicator has the following capabilities:

- Sound a tone device, which may be the PC or NC system speaker or another hardware device. In many cases the PC or NC speaker will not be available or will be in a position that is inaudible to the operator.
- Sound a two-tone indicator, providing simple pitch and volume control.
- Provide a synchronous one-shot indicator, similar to an Operating System’s Beep function.

Tone Indicator Class Diagram

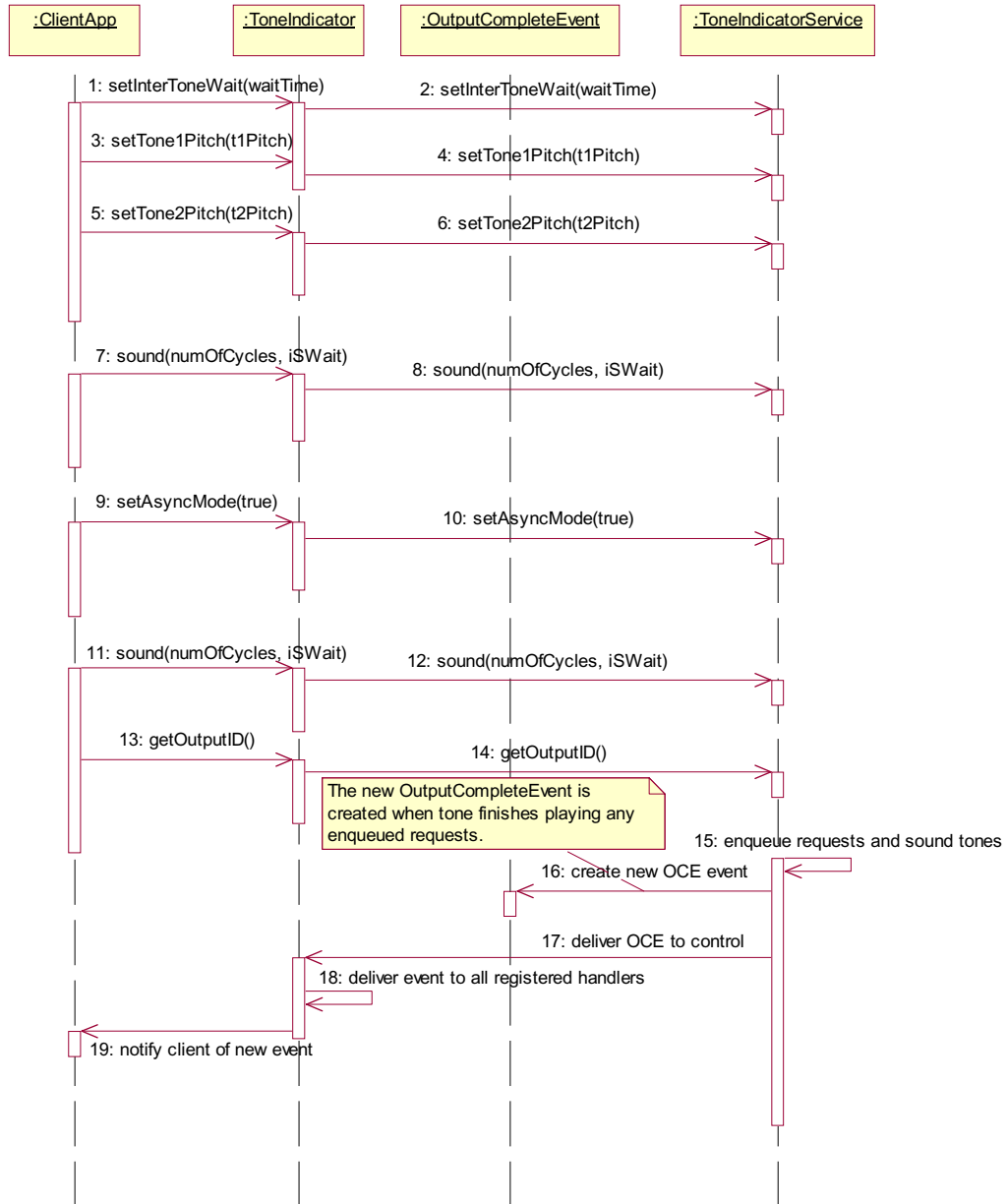
The following diagram shows the relationships between the Tone Indicator classes.



Tone Indicator Sequence Diagram *Added in Release 1.7*

The following sequence diagram shows the typical usage of the Tone Indicator device.

NOTE: we are assuming that the :ClientApp has already successfully opened and enabled the ToneIndicator device and registered its event handlers with the control. This means that the DeviceEnabled property is == true



Model

Updated in Release 1.7

The Tone Indicator device is for use when the POS hardware platform provides such capabilities external to the PC or NC standard speaker. Many POS systems have such devices, embedded, for example, in a keyboard, so that an indicator is always present at the point of sale.

This device supports a two-tone sound so that “*siren*” tones can be produced. The indicator is in general also started asynchronously so applications may perform other functions while waiting for the user to acknowledge the tone. There are also options to start the tone asynchronously with no count, so it runs forever, and be stopped by the application at a later time.

When the tone is started asynchronously, an **OutputCompleteEvent** is enqueued when all the tones have been played. This allows the application to know that the tone has stopped. For example, when the cash drawer is opened the tone could be started, quietly for a given number of cycles. If the cash drawer is closed then the tone is stopped explicitly by the application, if not then the notification by the **OutputCompleteEvent** allows the application to alter the prompt to the operator and possibly restart the tone a little louder.

The Tone Indicator follows the general device behavior model for output devices. Asynchronous output is handled as follows:

- The Device buffers the request in program memory, for delivery to the Physical Device as soon as the Physical Device can receive and process it, sets **OutputID** to an identifier for this request, and returns as soon as possible. When the request completes successfully, an **OutputCompleteEvent** is enqueued. A parameter of this event contains the **OutputID** of the completed request.

The **sound** method will not raise an exception due to a hardware problem. These errors will only be reported by an **ErrorEvent**. An exception will only be raised if the control is not claimed and enabled, a parameter is invalid, or the request cannot be enqueued. The first two error cases are due to an application error, while the last is a serious system resource exception.

- If an error occurs while performing an asynchronous request, an **ErrorEvent** is enqueued.
- Asynchronous output is performed on a first-in first-out basis.
- All buffered output data, including all asynchronous output, may be deleted by calling **clearOutput**. **OutputCompleteEvents** will not be delivered for cleared output. This method also stops any output that may be in progress (when possible).

Device Sharing

The Tone Indicator is a sharable device. Its device sharing rules are:

- After opening and enabling the device, the application may access all properties, methods, and enqueued **StatusUpdateEvents**.
- If more than one application has opened and enabled the device, each of these applications may access its properties and methods. **StatusUpdateEvents** will be delivered to all applications that are using the device and have registered to receive the event.
- If one application claims the tone indicator, then only that application may call **sound** and **soundImmediate**. Use of this feature will effectively restrict the tone indicator to the main application if that application claims the device at startup.
- The application that initiates asynchronous sounds is the only one that receives the corresponding **OutputCompleteEvents** and **ErrorEvents**.
- If a scenario exists such that an application is playing a sound and a separate application legally claims the device and plays a sound, then the sound being played from the first application will be interrupted. If the first application is in the midst of a synchronous **sound** method, an exception will be raised with the *ErrorCode* property set to *E_CLAIMED* from the method call. If the application has issued an asynchronous **sound** method, then no consistent reporting mechanism is possible and the first sound is simply terminated.
- See the “Summary” table for precise usage prerequisites.

Properties (UML attributes)

AsyncMode Property

Updated in Release 1.6

Syntax	AsyncMode: <i>boolean</i> { read-write, access after open }
Remarks	<p>If true, the sound method will be performed asynchronously. If false, tones are generated synchronously.</p> <p>This property is initialized to false when the device is first enabled following the open method. (In releases prior to 1.5, this description stated that initialization took place by the open method. In Release 1.5, it was updated for consistency with other devices.)</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

CapPitch Property

Syntax	CapPitch: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, the hardware tone generator has the ability to vary the pitch of the tone.</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 16.

CapVolume Property

Syntax	CapVolume: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, the hardware tone generator has the ability to vary the volume of the tone.</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 16.

InterToneWait Property

Updated in Release 1.6

Syntax	InterToneWait: <i>int32</i> { read-write, access after open }
Remarks	<p>Holds the number of milliseconds of silence between tone-1 and tone-2. If a gap is required after tone-2 but before a repeat of tone-1, then set the sound parameter <i>interSoundWait</i>.</p> <p>This property is initialized to zero when the device is first enabled following the open method. (In releases prior to 1.5, this description stated that initialization took place by the open method. In Release 1.5, it was updated for consistency with other devices.)</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

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

Value	Meaning
E_ILLEGAL	A negative value was specified.

Tone1Duration Property**Updated in Release 1.6**

Syntax	Tone1Duration: <i>int32</i> { read-write, access after open }
Remarks	<p>Holds the duration of the first tone in milliseconds. A value of zero or less will cause this tone not to sound.</p> <p>This property is initialized to zero when the device is first enabled following the open method. (In releases prior to 1.5, this description stated that initialization took place by the open method. In Release 1.5, it was updated for consistency with other devices.)</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

Tone1Pitch Property**Updated in Release 1.6**

Syntax	Tone1Pitch: <i>int32</i> { read-write, access after open }
Remarks	<p>Holds the pitch or frequency of the first tone in hertz. A value of zero or less will cause this tone not to sound.</p> <p>If the device does not support user-defined pitch (CapPitch is false), then any value greater than zero indicates that the tone indicator uses its default value.</p> <p>This property is initialized to zero when the device is first enabled following the open method. (In releases prior to 1.5, this description stated that initialization took place by the open method. In Release 1.5, it was updated for consistency with other devices.)</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

Tone1Volume Property**Updated in Release 1.6**

Syntax	Tone1Volume: <i>int32</i> { read-write, access after open }
Remarks	<p>Holds the volume of the first tone in percent of the device's capability, where 0 (or less) is silent and 100 (or more) is maximum.</p> <p>If the device does not support user-defined volume (CapVolume is false), then any value greater than zero indicates that the tone indicator uses its default value.</p> <p>This property is initialized to 100 when the device is first enabled following the open method. (In releases prior to 1.5, this description stated that initialization took place by the open method. In Release 1.5, it was updated for consistency with other devices.)</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

Tone2Duration Property***Updated in Release 1.6***

Syntax	Tone2Duration: <i>int32</i> { read-write, access after open }
Remarks	<p>Holds the duration of the second tone in milliseconds. A value of zero or less will cause this tone not to sound.</p> <p>This property is initialized to zero when the device is first enabled following the open method. (In releases prior to 1.5, this description stated that initialization took place by the open method. In Release 1.5, it was updated for consistency with other devices.)</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

Tone2Pitch Property***Updated in Release 1.6***

Syntax	Tone2Pitch: <i>int32</i> { read-write, access after open }
Remarks	<p>Holds the pitch or frequency of the second tone in hertz. A value of zero or less will cause this tone not to sound.</p> <p>If the device does not support user-defined pitch (CapPitch is false), then any value greater than zero indicates that the tone indicator uses its default value.</p> <p>This property is initialized to zero when the device is first enabled following the open method. (In releases prior to 1.5, this description stated that initialization took place by the open method. In Release 1.5, it was updated for consistency with other devices.)</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

Tone2Volume Property***Updated in Release 1.6***

Syntax	Tone2Volume: <i>int32</i> { read-write, access after open }
Remarks	<p>Holds the volume of the second tone in percent of the device's capability, where 0 (or less) is silent and 100 (or more) is maximum.</p> <p>If the device does not support user-defined volume (CapVolume is false), then any value greater than zero indicates that the tone indicator uses its default value.</p> <p>This property is initialized to 100 when the device is first enabled following the open method. (In releases prior to 1.5, this description stated that initialization took place by the open method. In Release 1.5, it was updated for consistency with other devices.)</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 16.

Methods (UML operations)

sound Method

Updated in Release 1.6

Syntax **sound (numberOfCycles: int32, interSoundWait: int32):
void { raises-exception, use after open-enable }**

Parameter	Description
<i>numberOfCycles</i>	The number of cycles to sound the indicator device. If UPOS_FOREVER, then start the indicator sounding and repeat continuously, else perform the sound for the specified number of cycles.
<i>interSoundWait</i>	When <i>numberOfCycles</i> is not one, then pause for <i>interSoundWait</i> milliseconds before repeating the tone cycle (before playing tone-1 again).

Remarks Sounds the indicator device, or start it sounding asynchronously.

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

The duration of an indicator cycle is:

Tone1Duration property +
InterToneWait property +
Tone2Duration property +
interSoundWait parameter (except on the last tone cycle)

After the tone indicator has started an asynchronous sound, then the sound may be stopped by using one of the following methods. (When a *numberOfCycles* value of UPOS_FOREVER was used to start the sound, then the application must use one of these to stop the continuous sounding of the tones.)

- **clearOutput**
- **soundImmediate**

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

Value	Meaning
E_CLAIMED	Indicates that another application has claimed the device and has taken over the tone device causing the sound from this method to be interrupted (can only be returned if AsyncMode is false.)
E_ILLEGAL	One of the following errors occurred: <ul style="list-style-type: none"> • <i>numberOfCycles</i> is neither a positive, non-zero value nor UPOS_FOREVER. • <i>numberOfCycles</i> is UPOS_FOREVER when AsyncMode is false. • A negative <i>interSoundWait</i> was specified • A negative InterToneWait was specified

soundImmediate Method

Syntax	soundImmediate (): void { raises-exception, use after open-enable }
Remarks	Sounds the hardware tone generator once, synchronously. Both tone-1 and tone-2 are sounded using InterToneWait . If asynchronous output is outstanding, then it is terminated before playing the immediate sound (as if clearOutput were called). This method is primarily intended for use in exception conditions when asynchronous output is outstanding, such as within an error event handler.
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 16.

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

See Also "Events" on page 15, **directIO** Method.

ErrorEvent**Updated in Release 1.7**

```

<< 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 device and a suitable response is necessary to process the error condition.

Attributes This event contains the following attributes:

Attributes	Type	Description
<i>ErrorCode</i>	<i>int32</i>	Error code causing the error event. See a list of Error Codes on page 16.
<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 property is settable). See values below.

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

Value	Meaning
EL_OUTPUT	Error occurred while processing asynchronous output.

The application's error processing may change *ErrorResponse* to one of the following values:

Value	Meaning
ER_RETRY	Retry the asynchronous output. The error state is exited. This is the default value.
ER_CLEAR	Clear all buffered output data, including all asynchronous output. The error state is exited.

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

See Also "Device Output Models" on page 21, "Device Information Reporting Model" on page 26, "Error Codes" on page 16

OutputCompleteEvent

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

Description Notifies the application that the queued output request associated with the *OutputID* property 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 21

StatusUpdateEvent

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

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

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 Tone Indicator device.

Note that Release 1.3 added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See "StatusUpdateEvent" on page 63.

Remarks Enqueued when the Tone Indicator device detects a power state change.

See Also "Events" on page 15

OLE for Retail POS — OPOS Implementation Reference

What Is “OLE for Retail POS?”

OLE for Retail POS provides an open device driver architecture that allows Point-of-Sale (“POS”)¹ hardware to be easily integrated into POS systems based on Microsoft Windows family of Operating Systems². It is an implementation of the UnifiedPOS Standard based upon the Microsoft Operating System Software and the OLE 2.x architecture.

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

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

Deliverables available for OPOS are:

- UnifiedPOS Programmer’s Guide – this document: For application developers and hardware providers.
- Header files with OPOS constants.
- No complete software components: Hardware providers or third-party providers develop and distribute these components.
- Reference Control Objects are available which incorporate the required functionality. These Control Objects, along with other helpful information may be found at the <http://www.nrf-arts.org> web site.

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

². Excludes Windows 3.x. Other future operating systems that support OLE Controls may also support OLE for Retail POS, depending upon software support by the hardware manufacturers or third-party developers.

Who Should Read This Section

This Section is targeted at an application developer who requires access to POS-specific peripheral devices and wishes to implement the UnifiedPOS Standard on a Microsoft Windows operating system platform. It is also targeted for the system developer who will write an OPOS Control, a vendor who wishes to write a OPOS Service Object, or an application developer who desires a better understanding of how to interface with OPOS under UnifiedPOS.

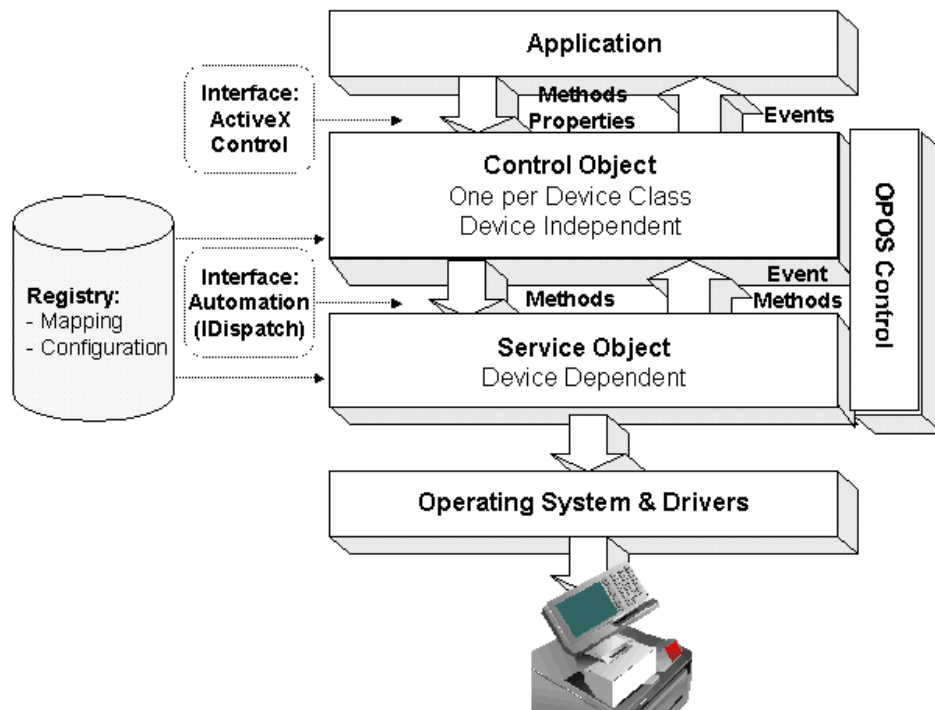
This guide assumes that the reader is familiar with the following:

- The UnifiedPOS Device chapters in this document.
- General characteristics of POS peripheral devices.
- ActiveX and Automation terminology and architecture.
- Familiarity with an ActiveX Control Container development environment, such as Microsoft Visual Basic or Microsoft Visual C++, will be useful.

General OLE for Retail POS Control Model

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

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



OPOS Definitions

Device Class

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

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

Control Object or CO

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

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

Service Object or SO

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

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

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

A Service Object is typically implemented as a local in-proc server (in a DLL). In theory, it may also be implemented as a local out-proc server (in a separate executable process). However, we have found that, in practice, out-proc servers do not work well for OPOS Service Objects, and do not recommend their use.

OPOS Control or Control

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

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

How an Application Uses an OPOS Control

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

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

Several applications may have an OPOS Control open at the same time. Therefore, after the device is opened, the application will often need to call the **ClaimDevice** method to gain exclusive access to the device. Many devices must be claimed before the Control allows access to its methods and properties. Claiming the device ensures that other applications do not interfere with the use of the device. The application may call the **ReleaseDevice** method when the device can be shared by other applications – for instance, at the end of a transaction.

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

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

In summary, the application follows this general sequence:

- **Open** method: Call to link the Control Object to the Service Object.
- **ClaimDevice** method: Call to gain exclusive access to the device. Required for exclusive-use devices; optional for some sharable devices. (See “Device Sharing Model”, page A-10 for more information).
- **DeviceEnabled** property: Set to TRUE to make the device operational. (For sharable devices, the device may be enabled without first claiming it.)
- Use the device.
- **DeviceEnabled** property: Set to FALSE to disable the device.
- **ReleaseDevice** method: Call to release exclusive access to the device.
- **Close** method: Call to release the Service Object from the Control Object.

When Methods and Properties May Be Accessed

Methods

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

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

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

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

Properties

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

The following properties are always initialized:

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

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

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

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

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

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

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

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

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

Status, Result Code, and State Model

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

Name	Meaning
State	A property containing the current state of the Control: OPOS_S_CLOSED OPOS_S_IDLE OPOS_S_BUSY OPOS_S_ERROR
ResultCode	A property containing the status of the most recent method or the most recently changed writable property: OPOS_SUCCESS OPOS_E_CLOSED OPOS_E_CLAIMED OPOS_E_NOTCLAIMED OPOS_E_NOSERVICE OPOS_E_DISABLED OPOS_E_ILLEGAL OPOS_E_NOHARDWARE OPOS_E_OFFLINE OPOS_E_NOEXIST OPOS_E_EXISTS OPOS_E_FAILURE OPOS_E_TIMEOUT OPOS_E_BUSY OPOS_E_EXTENDED
ResultCodeExtended	A property containing the extended status of the most recent method or the most recently changed writable property. Value varies by ResultCode and by device class.
StatusUpdateEvent	An event fired when some class-specific state or status variable has changed. Release 1.3 and later: All devices may be able to report device power state. See "Device Power Reporting Model" on page A-17.
ErrorEvent	An event fired when the State is changed to Error.

Status Model

The rules of the status model are as follows:

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

Result Code Model

The rules of the result code model are as follows:

- Every method returns a result code. This code is also placed into **ResultCode**.
- Setting a writable property causes a result code to be placed into **ResultCode**.
- The **ResultCode** OPOS_SUCCESS is assigned the value of zero. Non-zero values indicate an error or warning.
- The Control must select one of the result codes listed below. If the Control sets **ResultCode** to OPOS_E_EXTENDED, then it must set **ResultCodeExtended** to one of the values specified in the device class documentation. (That is, when this **ResultCode** value is selected, then **ResultCodeExtended** may only contain one of the values listed in this document for the device class, in the appropriate method or property section.)

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

State Model

Updated in Release 1.7

The rules of the state model are as follows:

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

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

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

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

- Retry – If the locus is Output: Retry the asynchronous output and exit the error state. If an error occurs while retrying, then another **ErrorEvent** will be generated.
If the locus is Input: Some devices support retrying the input, if retry can be controlled by the Service Object.
“Retry” is the default response when the locus is “Output.”
- Clear – Clear all buffered output data (including all asynchronous output) or buffered input data and exit the error state.
“Clear” is the default response when the locus is “InputNoData.”
- Continue – Use only if the locus is InputWithData. This response acknowledges the error and directs the Control to continue processing. The Control remains in the error state, and will deliver additional data events as directed by the **DataEventEnabled** property. When all input has been delivered and the **DataEventEnabled** property is again set to TRUE, then another **ErrorEvent** is delivered with locus “InputNoData.”
“Continue” is the default response when the locus is “InputNoData.”

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

Device Sharing Model

The OLE for Retail POS device sharing model supports devices that are to be used exclusively by one application³ at a time, as well as devices that may be partially or fully shared by multiple applications. (See “When Methods and Properties May Be Accessed”, page A-5, for other details.) All OPOS Controls may be opened by more than one application at a given time. Some or many of the activities that an application can perform with the Control, however, may be restricted to an application that claims access to the device.

Exclusive-Use Devices

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

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

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

Sharable Devices

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

Note:

One might argue that all devices should be defined as sharable to allow maximum flexibility to applications. In practical use, this flexibility is unlikely to be useful. The downside is an implementation that may be significantly more complex and less likely to be accurate.

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

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

Events

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

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

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

- The application thread is busy processing other messages. OPOS Controls are to follow the OLE Apartment Threading model. According to OLE Apartment Threading rules, events are to be delivered on the thread that created the COM object, which will usually be the application's main thread. If the application is processing another message, then event delivery must wait until this processing has finished.
- The application has set the property **FreezeEvents** to TRUE.
- The event type is **DataEvent** or an input **ErrorEvent**, but the property **DataEventEnabled** is FALSE. (See "Input Model" on page A-14.)

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

Note – Terminology

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

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

Rules on the management of the queue of events are:

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

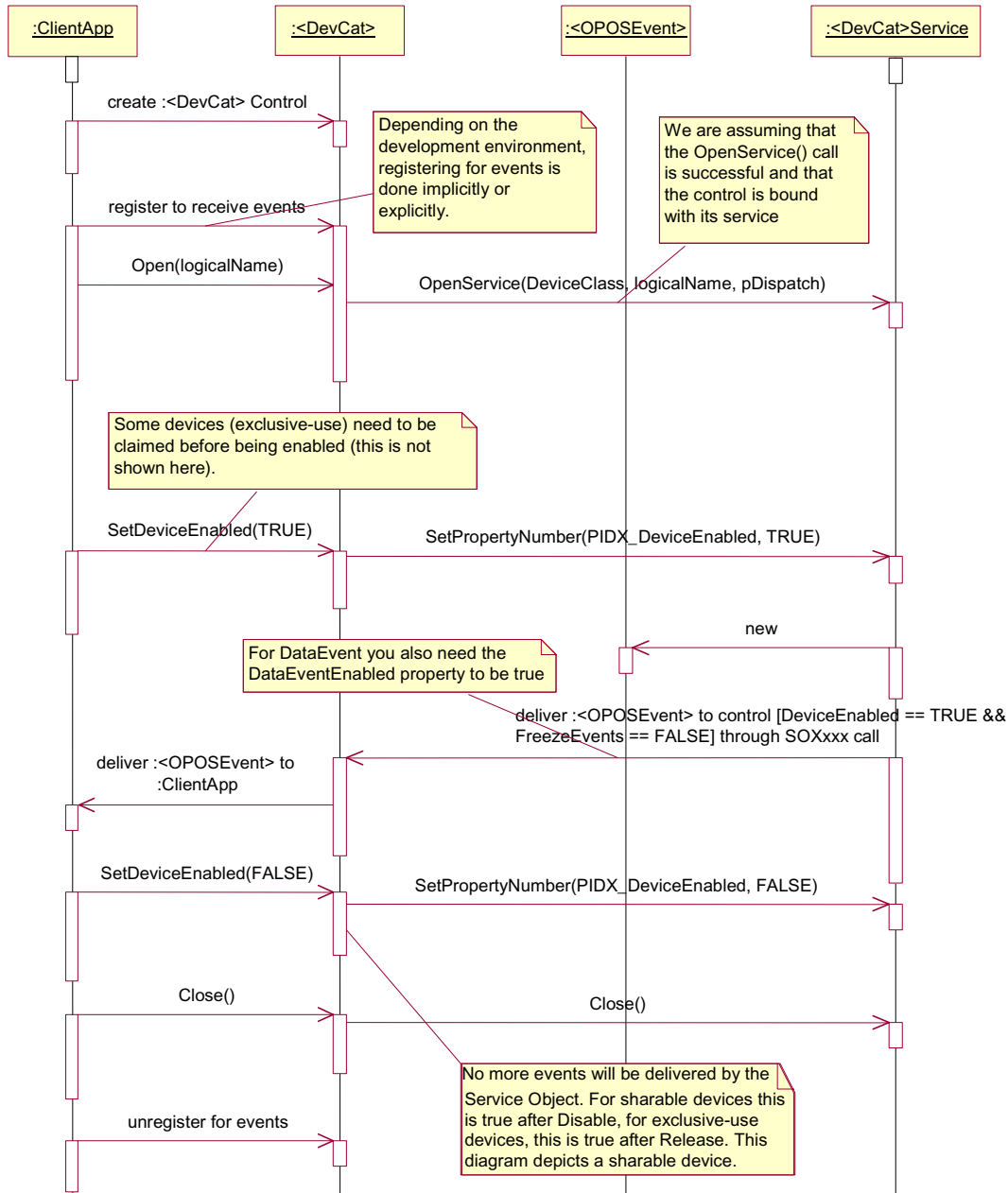
While within an event handler, the application may access properties and call methods. However, the application must not call the **ReleaseDevice** or **Close** methods from an event handler, since **ReleaseDevice** may shut down event handling (possibly including a thread that caused the event to be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

OPOS Event Registration Sequence Diagram

Added in Release 1.7

The following sequence diagram depicts the typical OPOS event registration process.

NOTE: this diagram shows the typical event registration process for a Service Object in OPOS. Various details are omitted on purpose to make the diagram clearer. Also, DevCat == POSPrinter, CashDrawer, ... and other UnifiedPOS device categories.



Input Model

The OLE for Retail POS input model supports event-driven input. Event-driven input allows input data to be received after **DeviceEnabled** is set to TRUE. Received data is enqueued as a **DataEvent**, which is delivered to the application when preconditions are correct. If the **AutoDisable** property is TRUE when data is received, then the control will automatically disable itself, setting **DeviceEnabled** to FALSE. This will inhibit the Control from 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 Control enqueues and delivers a **DataEvent**. (If input has already been enqueued, the **DataEvent** will be delivered.) This event may include input status information through a numeric parameter. The Control places the input data plus other information as needed into device specific-specific properties just before the event is fired.

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

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

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

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

- **InputWithData** (OPOS_EL_INPUT_DATA) – Only enqueued if the error occurred while one or more **DataEvents** are enqueued. It is enqueued ahead of all **DataEvents**. (A typical implementation would place it at the head of the event queue.) This event gives the application the ability to immediately clear the input, or to optionally alert the user to the error and process the buffered input.

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

- **InputNoData** (OPOS_EL_INPUT) – Delivered when an error has occurred and there is no data available. (A typical implementation would place it at the tail of the event queue.) If some input data was already enqueued when the error occurred, then an **ErrorEvent** with the locus “InputWithData” was enqueued and delivered first, and then this error event is delivered after all **DataEvents** have been fired. (If an “InputWithData” event was delivered and the application event handler responded with a “Clear”, then this “InputNoData” event is not delivered.)

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

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

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

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

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

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

Output Model

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

Synchronous Output

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

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

Asynchronous Output

Updated in Release 1.7

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

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

If an error occurs while performing an asynchronous request, an **ErrorEvent** is fired. The application's event handler can either retry the outstanding output or clear it. The Control is in the Error state while the **ErrorEvent** is in progress. (Note that if the condition causing the error was not corrected, then the Control may immediately reenter the Error state and fire another **ErrorEvent**.)

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

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

Device Power Reporting Model

Added in OPOS Release 1.3, Updated in Release 1.8

Applications frequently need to know the power state of the devices they use. Earlier versions of OPOS had no consistent method for reporting this information. **Note:** This model is not intended to report PC or POS Terminal power conditions (such as “on battery” and “battery low”). Reporting of these conditions is now managed by the POSPower device category, see page 601.

Model

OPOS segments device power into three states:

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

In addition, one combination state is defined:

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

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

Note – Enabled/Disabled vs. Power States

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

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

Properties

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

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

Power Reporting Requirements for DeviceEnabled

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

- When the Control changes from **DeviceEnabled** FALSE to TRUE, then begin monitoring the power state:

- If the device is ONLINE, then:

PowerState is set to OPOS_PS_ONLINE.

A **StatusUpdateEvent** is fired with *Status* parameter set to OPOS_SUE_POWER_ONLINE.

- If the device power state is OFF, OFFLINE, or OFF_OFFLINE, then the Control may choose to fail the enable, setting **ResultCode** to OPOS_E_NOHARDWARE or OPOS_E_OFFLINE.

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

PowerState is set to OPOS_PS_OFF, OPOS_PS_OFFLINE, or OPOS_PS_OFF_OFFLINE.

A **StatusUpdateEvent** is fired with *Status* parameter set to OPOS_SUE_POWER_OFF, OPOS_SUE_POWER_OFFLINE, or OPOS_SUE_POWER_OFF_OFFLINE.

- When the Control changes from **DeviceEnabled** TRUE to FALSE, then OPOS assumes that the Control is no longer monitoring the power state. Therefore:

PowerState is set to OPOS_PS_UNKNOWN.

Device Information Reporting Model *Added in Release 1.8.*

POS Applications, as well as System Management agents, frequently need to monitor the current configuration and usage metrics of the various POS devices that are attached to the POS terminal.

Examples of configuration data are the device's *Serial Number*, *Firmware Version*, and *Connection Type*. Examples of usage data for the POSPrinter device are the *Number of Lines Printed*, *Number of Hours Running*, *Number of paper cuts*, etc. Examples of usage data for the Scanner device are the *Number of scans*, *Number of Hours Running*, etc. Examples of usage data for the MSR device are the *Number of successful swipes*, *Number of swipes resulting in errors*, *Number of Hours Running*, etc. See page 27 for examples of XML definitions of the device statistics accumulated per POS device category.

In some cases, the data may be accumulated and stored within the device itself. In other cases, the data may be accumulated by the Service and stored, possibly on the POS terminal or store controller.

In order for multiple applications (for example a POS application and a System Management application) to obtain statistics from the same device, proper care must be taken by both applications so that the device can be made accessible when required. This is done by using the **ClaimDevice** method and by setting **DeviceEnabled** to TRUE when access to a device is required and then setting **DeviceEnabled** to FALSE and using the **ReleaseDevice** method when access to the device is no longer needed. Coordination of device access via this mechanism is the responsibility of the applications themselves.

Statistics Reporting Properties and Methods

The UnifiedPOS device information reporting model adds the following common properties and methods across all device classes.

- **CapStatisticsReporting** property. Identifies the reporting capabilities of the device. When **CapStatisticsReporting** is FALSE, then no statistical data regarding the device is available. This is equivalent to Services compatible with prior versions of the specification. When **CapStatisticsReporting** is TRUE, then some statistical data for the device is available.
- **CapUpdateStatistics** property. Defines whether gathered statistics (or some of them) can be reset/updated by the application. This property is only valid if **CapStatisticsReporting** is TRUE. When **CapUpdateStatistics** is FALSE, then none of the statistical data can be reset/updated by the application. Otherwise, when **CapUpdateStatistics** is TRUE, then (some of) the statistical data can be reset/updated by the application.
- **ResetStatistics** method. Can only be called if both **CapStatisticsReporting** and **CapUpdateStatistics** are TRUE. This method resets one, some, or all of the resettable device statistics to zero.
- **RetrieveStatistics** method. Can only be called if **CapStatisticsReporting** is TRUE. This method retrieves one, some, or all of the accumulated statistics for the device.
- **UpdateStatistics** method. Can only be called if both **CapStatisticsReporting** and **CapUpdateStatistics** are TRUE. This method updates one, some, or all of the resettable device statistics to the supplied values.

OPOS Component Descriptions

The following sections are arranged as follows and provide detailed information on how an Application is expected to interface with a device covered under OPOS.

Section 1:

Describes the specific characteristics of the data types that OPOS uses as they relate to the Windows OPOS implementation.

Section 2:

Provides interface descriptions for the properties, methods, and events specific to OPOS. For thorough description of these, one should consult the applicable chapters located in this document.

Section 3:

Details the OPOS use of the system registry specific to Windows.

Section 4:

Contains the list of the C++ OPOS application header files.

Section 5:

Provides some miscellaneous additional technical information to help the Application Developer understand some of the finer details of a Windows OPOS implementation.

Section 6:

Provides additional information on **ClaimDevice** and **ReleaseDevice** methods which became necessary as a result of Microsoft's ActiveX changes that affected the **Claim** and **Release** method naming convention that was used in OPOS 1.4 and earlier editions.

Section 7:

Provides the Change History previously contained in the OPOS Application Programmer's Guide (OPOS APG).

Section 8:

Provides information previously contained in the OPOS Control Programmer's Guide (OPOS CPG). Targeted at system developers who intend to write an OPOS Control.

Section 1: OPOS Data Types

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

Type	Meaning
BOOL	An integer with the legal values TRUE (non-zero) and FALSE (zero). COM IDL type: VARIANT_BOOL (short). Values VARIANT_TRUE (-1) and VARIANT_FALSE (0). VARIANT type: VT_BOOL
BSTR	A character string. Consists of a length component followed by the string and a terminating NUL (0) character. See “System Strings (BSTR)” (page A-73) for more information. COM IDL type: BSTR (unsigned short*) VARIANT type: VT_BSTR
BSTR*	A pointer to a character string. COM IDL type: BSTR* (unsigned short**) VARIANT type: VT_BYREF VT_BSTR
LONG	An integer with a size of 32 bits. COM IDL type: long VARIANT type: VT_I4
LONG*	A pointer to a 32-bit integer. COM IDL type: long* VARIANT type: VT_BYREF VT_I4
CURRENCY	Release 1.3 and later A monetary value. An integer with a size of 64 bits. The value assumes four decimal places. For example, if the integer is “1234567”, then the value is “123.4567”. COM IDL type: CURRENCY (union tagCY) “union tagCY” is declared as <pre>{ struct { long Hi; long Lo; }; __int64 int64; };</pre> VARIANT type: VT_CY
CURRENCY*	Release 1.3 and later A pointer to a CURRENCY value. COM IDL type: CURRENCY* (union tagCY*) VARIANT type: VT_BYREF VT_CY

Section 2: OPOS Interface Descriptions

Information in this section further defines the requirements of the UnifiedPOS for a Windows OS environment implementation. The common Properties, Methods, and Events are included to help transition from the UML given in Chapter 1 to the specifics for the Windows environment.

Next, tables are included that outline the specific programmatic examples for each of the device classifications and reference back to the UML for the respective devices.

The examples have been provided in Visual Basic and Visual C++ as the Windows OS reference programming platforms. Other programming languages written for the Windows OS environment may be supported as long as they comply to the Microsoft OLE 2.x.

OPOS Common Properties, Methods, and Events

Common Properties

Updated in Release 1.8

OPOS implementation specific definitions of the Common Properties.

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 }	1.2	1
BinaryConversion	<i>Long</i>	{ read-write }	1.2	
CapPowerReporting	<i>Long</i>	{ read-only }	1.3	
CapStatisticsReporting	<i>Boolean</i>	{ read-only }	1.8	
CapUpdateStatistics	<i>Boolean</i>	{ read-only }	1.8	
CheckHealthText	<i>String</i>	{ read-only }	1.0	
Claimed	<i>Boolean</i>	{ read-only }	1.0	
DataCount	<i>Long</i>	{ read-only }	1.2	1
DataEventEnabled	<i>Boolean</i>	{ read-write }	1.0	1
DeviceEnabled:	<i>Boolean</i>	{ read-write }	1.0	
FreezeEvents	<i>Boolean</i>	{ read-write }	1.0	
OpenResult	<i>Long</i>	{ read-only }	1.5	
OutputID	<i>Long</i>	{ read-only }	1.0	2
PowerNotify	<i>Long</i>	{ read-write }	1.3	
PowerState	<i>Long</i>	{ read-only }	1.3	
ResultCode	<i>Long</i>	{ read-only }	1.0	
ResultCodeExtended	<i>Long</i>	{ read-only }	1.0	
State	<i>Long</i>	{ read-only }	1.0	
ControlObjectDescription	<i>String</i>	{ read-only }	1.0	
ControlObjectVersion	<i>Long</i>	{ read-only }	1.0	
ServiceObjectDescription	<i>String</i>	{ read-only }	1.0	
ServiceObjectVersion	<i>Long</i>	{ read-only }	1.0	
DeviceDescription	<i>String</i>	{ read-only }	1.0	
DeviceName	<i>String</i>	{ read-only }	1.0	

Usage Notes:

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

Common Methods

Added in Release 1.8

OPOS implementation specific definitions of the Common Methods.

Methods (UML operations)

<i>Name</i>	<i>Version</i>
LONG Open (BSTR DeviceName);	1.0
LONG Close ();	1.0
LONG ClaimDevice^a (LONG Timeout);	1.0
LONG ReleaseDevice^a ();	1.0
LONG CheckHealth (LONG Level);	1.0
LONG ClearInput ();	1.0
LONG ClearOutput ();	1.0
LONG DirectIO (LONG Command, LONG* pData, BSTR* pString);	1.0
LONG ResetStatistics (BSTR StatisticsBuffer);	1.8
LONG RetrieveStatistics (BSTR* pStatisticsBuffer);	1.8
LONG UpdateStatistics (BSTR StatisticsBuffer);	1.8

- 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 name used by Microsoft's Component Object Model (COM).

OPOS Programmatic Names

OPOS implementation specific definitions of the POS Device Categories' programmatic IDs.

UnifiedPOS Device Programmatic Names	OPOS Programmatic IDs
BumpBar	OPOS.BumpBar
CashChanger	OPOS.CashChanger
CashDrawer	OPOS.CashDrawer
CAT	OPOS.CAT
CheckScanner	OPOS.CheckScanner
CoinDispenser	OPOS.CoinDispenser
FiscalPrinter	OPOS.FiscalPrinter
HardTotals	OPOS.HardTotals
Keylock	OPOS.Keylock
LineDisplay	OPOS.LineDisplay
MICR	OPOS.MICR
MotionSensor	OPOS.MotionSensor
MSR	OPOS.MSR
PINPad	OPOS.PINPad
PointCardRW	OPOS.PointCardRW
POSKeyboard	OPOS.POSKeyboard
POSPower	OPOS.POSPower
POSPrinter	OPOS.POSPrinter
RemoteOrderDisplay	OPOS.RemoteOrderDisplay
Scale	OPOS.Scale
Scanner	OPOS.Scanner
SignatureCapture	OPOS.SignatureCapture
SmartCardRW	OPOS.SmartCardRW
ToneIndicator	OPOS.ToneIndicator

Properties

AutoDisable Property R/W

Added in Release 1.2

Syntax **BOOL AutoDisable;**

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

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

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

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

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

Value	Meaning
OPOS_SUCCESS	The property was set successfully.

BinaryConversion Property R/W**Added in Release 1.2****Syntax** LONG BinaryConversion;

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

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

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

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

“In the OPOS environment, the format of this data depends upon the value of the **BinaryConversion** property. See **BinaryConversion** property on page A-28.”

The following table defines the affected device categories and affected Properties and Methods.

Device Category	Property/Method/Event Name	Reference Page
Common PME	DirectIOEvent	See page 60.
CAT	AdditionalSecurityInformation	See page 146.
CheckScanner	ImageData	See page 196.
FiscalPrinter	PrintNormal	See page 314.
HardTotals	Read Write	See page 386. See page 390.
LineDisplay	DefineGlyph DisplayText DisplayTextAt	See page 436. See page 440. See page 442.
PINPad	ComputeMAC (2 parameters)	See page 531.
PointCardRW	PrintWrite ValidateData	See page 581. See page 583.
POSPrinter	PrintBarCode PrintImmediate PrintNormal PrintTwoNormal (2 parameters) SetLogo ValidateData	See page 701. See page 708. See page 710. See page 712. See page 718. See page 721.
RemoteOrderDisplay	DisplayData	See page 758.
Scale	DisplayText	See page 785.
Scanner	ScanData ScanDataLabel	See page 798. See page 799.
SignatureCapture	PointArray RawData	See page 815. See page 816.

The binary conversion values are:

Value	Meaning
OPOS_BC_NONE	Data is placed one byte per BString character, with no conversion. (This is the default, and is the behavior of OPOS Service Objects prior to 1.2.)
OPOS_BC_NIBBLE	Each byte is converted into two characters. (This option provides for the fastest conversion between binary and ASCII characters.) Each data byte is converted as follows: First character = 0x30 + bits 7-4 of the data byte. Second character = 0x30 + bits 3-0 of the data byte. Example: Byte value 154 = 0x9A is converted into the characters 0x39 0x3A (= the string "9:"). Note that this conversion is not the more common hexadecimal ASCII, which would have converted 154 to 0x39 0x41 (= the string "9A").
OPOS_BC_DECIMAL	Each byte is converted into three characters. (This option provides for the easiest conversion between binary and ASCII characters for Visual Basic and similar languages.) VAL(<i>string</i>) may be used on each 3 characters to convert from ASCII to binary. RIGHT("^^"+STR(<i>byte</i>), 3) may be used to produce 3 ASCII characters from each byte, where '^' represents the space character. Example 1: Byte value 154 = 0x9A becomes the characters 0x31 0x35 0x34 (= the string "154"). Example 2: Byte value 8 becomes the characters 0x30 0x30 0x38 (= the string "008"). Requirements for a Service Object are: (1) When the Service Object converts from ASCII to binary, it must allow either leading spaces or ASCII zeroes, since STR(<i>byte</i>) produces a leading space. (For example, the application may pass "^^8^27", where '^' represents the space character, which will be interpreted as the two bytes 8 (0x08) and 27 (0x1B).) (2) When the Service Object converts from binary to ASCII, it must always convert each byte into exactly three ASCII decimal characters (range 0x30 to 0x39).

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

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

To better understand the “direction” of the conversion, determine if the data flow follows the Output Model or the Input Model. If the flow follows the Output Model, then the application must adhere to the first responsibility listed above. If the flow follows the Input Model, then the application must adhere to the second responsibility listed above.

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

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

Value	Meaning
OPOS_SUCCESS	The property was set successfully.
OPOS_E_ILLEGAL	An illegal value was specified.

CapPowerReporting Property

Added in Release 1.3

Syntax LONG CapPowerReporting;

Remarks Identifies the reporting capabilities of the device.

The power reporting values are:

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

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

CapStatisticsReporting Property***Added in Release 1.8***

- Syntax** **BOOL CapStatisticsReporting;**
- Remarks** If TRUE, the device accumulates and can provide various statistics regarding usage; otherwise no usage statistics are accumulated. The information accumulated and reported is device specific, and is retrieved using the **RetrieveStatistics** method.
- This property is initialized by the **Open** method.
- See Also** **RetrieveStatistics** Method.

CapUpdateStatistics Property***Added in Release 1.8***

- Syntax** **BOOL CapUpdateStatistics;**
- Remarks** If TRUE, the device statistics, or some of the statistics, can be reset to zero using the **ResetStatistics** method, or updated using the **UpdateStatistics** method.
- If **CapStatisticsReporting** is FALSE, then **CapUpdateStatistics** is also FALSE.
- This property is initialized by the **Open** method.
- See Also** **CapStatisticsReporting** Property, **ResetStatistics** Method, **UpdateStatistics** Method.

CheckHealthText Property

- Syntax** **BSTR CheckHealthText;**
- Remarks** Holds the results of the most recent call to the **CheckHealth** method. The following examples illustrate some possible diagnoses:
- “Internal HCheck: Successful”
 - “External HCheck: Not Responding”
 - “Interactive HCheck: Complete”
- Before the first **CheckHealth** method call, its value is uninitialized.

Claimed Property

- Syntax** **BOOL Claimed;**
- Remarks** If TRUE, the device is claimed for exclusive access.
 If FALSE, the device is released for sharing with other applications.
- Many devices must be claimed before the Control will allow access to many of its methods and properties, and before it will fire events to the application.
- The value of **Claimed** is initialized to FALSE by the **Open** method.

ControlObjectDescription Property

Syntax **BSTR ControlObjectDescription;**

Remarks String identifying the Control Object and the company that produced it.
The property identifies the Control Object. A sample returned string is:

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

This property is always readable.

ControlObjectVersion Property

Syntax **LONG ControlObjectVersion;**

Remarks Control Object version number.

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

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

A sample version number is:

```
    1002038
```

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

This property is always readable.

Note:

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

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

- Reading an unsupported property: The Control Object returns the property's uninitialized value. (See "When Methods and Properties May Be Accessed" on page A-5 for uninitialized property default values.)
 - Writing an unsupported property: The Control Object returns, but must remember that an unsupported property write or method call occurred. Then, if the application reads the **ResultCode** property, the Control Object must return a value of OPOS_E_NOSERVICE (rather than reading the current **ResultCode** from the Service Object). It must do this until the next property write or method call, at which time **ResultCode** is set by that API.
 - Calling an unsupported method: The Control Object returns a value of OPOS_E_NOSERVICE, and must remember that an unsupported property write or method call occurred. Then, if the application reads the **ResultCode** property, the Control Object must return a value of OPOS_E_NOSERVICE (rather than reading the current **ResultCode** from the Service Object). It must do this until the next property write or method call, at which time **ResultCode** is set by that API.
-

DataCount Property***Added in Release 1.2*****Syntax** **LONG DataCount;****Remarks** Holds the number of enqueued **DataEvents** at the control.

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

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

DataEventEnabled Property R/W**Syntax** **BOOL DataEventEnabled;**

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

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

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

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

Value	Meaning
OPOS_SUCCESS	The property was set successfully.

DeviceDescription Property**Syntax** **BSTR DeviceDescription;****Remarks** String identifying the device.

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

“NCR 7192-0184 Printer, Japanese Version”

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

DeviceEnabled Property R/W

Syntax	BOOL DeviceEnabled;								
Remarks	<p>When TRUE, the device has been placed in an operational state. If changed to TRUE, then the device is brought to an operational state.</p> <p>When FALSE, the device has been disabled. If changed to FALSE, then the device is physically disabled when possible, any subsequent input will be discarded, and output operations are disallowed.</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>Release 1.3 and later: The device's power state may be reported while DeviceEnabled is TRUE.</p> <p>This property is initialized to FALSE by the Open method.</p>								
Return	<p>When this property is set, one of the following values is placed in the ResultCode property:</p> <table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>OPOS_SUCCESS</td> <td>The property was set successfully.</td> </tr> <tr> <td>OPOS_E_NOTCLAIMED</td> <td>An exclusive use device must be claimed before the device may be enabled.</td> </tr> <tr> <td><i>Other Values</i></td> <td>See ResultCode.</td> </tr> </tbody> </table>	Value	Meaning	OPOS_SUCCESS	The property was set successfully.	OPOS_E_NOTCLAIMED	An exclusive use device must be claimed before the device may be enabled.	<i>Other Values</i>	See ResultCode .
Value	Meaning								
OPOS_SUCCESS	The property was set successfully.								
OPOS_E_NOTCLAIMED	An exclusive use device must be claimed before the device may be enabled.								
<i>Other Values</i>	See ResultCode .								

DeviceName Property

Syntax	BSTR DeviceName;
Remarks	<p>Short string identifying the device.</p> <p>The property identifies the device and any pertinent information about it. This is a short version of DeviceDescription and should be limited to 30 characters.</p> <p>DeviceName will typically be used to identify the device in an application message box, where the full description is too verbose. A sample returned string is:</p> <p style="padding-left: 40px;">"NCR 7192 Printer, Japanese"</p> <p>This property is initialized by the Open method.</p>

FreezeEvents Property R/W**Syntax** **BOOL FreezeEvents;****Remarks** When TRUE, the application has requested that the Control not deliver events. Events will be held by the Control until events are unfrozen.

When FALSE, the application allows events to be delivered. If some events have been held while events were frozen and all other conditions are correct for delivering the events, then changing **FreezeEvents** to FALSE will cause these events to be delivered.⁴

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

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

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

Value	Meaning
OPOS_SUCCESS	The property was set successfully.

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

OpenResult Property***Added in Release 1.5*****Syntax** **LONG OpenResult;****Remarks** Holds additional details about the most recent **Open** method.

The open result values are:

Value	Meaning
OPOS_SUCCESS	Successful open.
OPOS_OR_ALREADYOPEN	Control already open.
OPOS_OR_REGBADNAME	The registry does not contain a key for the specified device name.
OPOS_OR_REGPROGID	Could not read the device name key's default value, or could not convert the Programmatic ID it holds into a valid Class ID.
OPOS_OR_CREATE	Could not create a service object instance, or could not get its IDispatch interface.
OPOS_OR_BADIF	The service object does not support one or more of the methods required by its release.
OPOS_OR_FAILEDOPEN	The service object returned a failure status from its open call, but does not have a more specific failure code.
OPOS_OR_BADVERSION	The service object major version number does not match the control object major version number. The following values can be returned by the Service Object if it returns a failure status from its open call. The Service Object can choose to return one of these, if applicable, or define additional values. (See the Control Programmer's Guide's GetOpenResult description for details on how the Service Object returns these values. If the Service Object does not implement GetOpenResult, then OpenResult returns OPOS_OR_FAILEDOPEN.)
OPOS_OR_NOPORT	The Service Object tried to access an I/O port (for example, an RS232 port) during Open processing, but the port that is configured for the DeviceName is invalid or inaccessible.

As a general rule, an SO should refrain from accessing the physical device until the DeviceEnabled property is set to TRUE. But in some cases, it may require some access at Open; for instance, to dynamically determine the device type in order to set the DeviceName and DeviceDescription properties.

OPOS_OR_S_NOTSUPPORTED

The Service Object does not support the specified device.

The SO has determined that it does not have the ability to control the device it is opening. This determination may be due to an inspection of the registry entries for the device, or dynamic querying of the device during open processing.

OPOS_OR_S_CONFIG Configuration information error.

Usually this is due to incomplete configuration of the registry, such that the SO does not have sufficient or valid data to open the device.

OPOS_OR_S_SPECIFIC Errors greater than this value are service object-specific.

If the previous return values do not apply, then the SO may define additional OpenResult values. These values are Service Object-specific, but may be of value in these cases:

- 1) The Application logs or reports this error during debug and testing.
- 2) The Application adds SO-specific logic, to attempt to report more error conditions or to recover from them.

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

OutputID Property

Syntax **LONG OutputID;**

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

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

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

PowerNotify Property R/W**Added in Release 1.3****Syntax** **LONG PowerNotify;**

Remarks Contains the type power notification selection made by the Application.
 The power notification values are:

Value	Meaning
OPOS_PN_DISABLED	The Control will not provide any power notifications to the application. No power notification StatusUpdateEvents will be fired, and PowerState may not be set.
OPOS_PN_ENABLED	The Control 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 to OPOS_PN_DISABLED by the **Open** method. This value provides compatibility with earlier releases.

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

Value	Meaning
OPOS_SUCCESS	The property was set successfully.
OPOS_E_ILLEGAL	One of the following occurred: The device is already enabled. PowerNotify = OPOS_PN_ENABLED but CapPowerReporting = OPOS_PR_NONE.
<i>Other Values</i>	See ResultCode .

PowerState Property**Added in Release 1.3****Syntax** **LONG PowerState;**

Remarks Contains the current power condition, if it can be determined.
 The power reporting values are:

Value	Meaning
OPOS_PS_UNKNOWN	Cannot determine the device's power state, for one of the following reasons: CapPowerReporting = OPOS_PR_NONE. Device does not support power reporting.

	PowerNotify = OPOS_PN_DISABLED. Power notifications are disabled.
	DeviceEnabled = FALSE. Power state monitoring does not occur until the device is enabled.
OPOS_PS_ONLINE	The device is powered on and ready for use. Can be returned if CapPowerReporting = OPOS_PR_STANDARD or OPOS_PR_ADVANCED.
OPOS_PS_OFF	The device is off or detached from the terminal. Can only be returned if CapPowerReporting = OPOS_PR_ADVANCED.
OPOS_PS_OFFLINE	The device is powered on but is either not ready or not able to respond to requests. Can only be returned if CapPowerReporting = OPOS_PR_ADVANCED.
OPOS_PS_OFF_OFFLINE	The device is either off or offline. Can only be returned if CapPowerReporting = OPOS_PR_STANDARD.

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

ResultCode Property

Syntax	LONG ResultCode;
Remarks	<p>This property is set by each method. It is also set when a writable property is set.</p> <p>This property is always readable. Before the Open method is called, it returns the value OPOS_E_CLOSED.</p> <p>It is conceivable that more than one of the following result codes could be valid for a particular failure. The order of error reporting precedence for such scenarios is the following:</p> <ul style="list-style-type: none"> • OPOS_E_CLAIMED • OPOS_E_NOTCLAIMED • OPOS_E_DISABLED

The result code values are:

Value	Meaning
OPOS_SUCCESS	Successful operation.
OPOS_E_CLOSED	Attempt was made to access a closed device.
OPOS_E_CLAIMED	Attempt was made to access a device that is claimed by another process. The other process must release the device before this access may be made. For exclusive-use devices, the application will also need to claim the device before the access is legal.
OPOS_E_NOTCLAIMED	Attempt was made to access an exclusive-use device that must be claimed before the method or property set action can be used. If the device is already claimed by another process, then the status OPOS_E_CLAIMED is returned instead.
OPOS_E_NOSERVICE	The Control cannot communicate with the Service Object. Most likely, a setup or configuration error must be corrected.
OPOS_E_DISABLED	Cannot perform operation while device is disabled.
OPOS_E_ILLEGAL	Attempt was made to perform an illegal or unsupported operation with the device, or an invalid parameter value was used.
OPOS_E_NOHARDWARE	The device is not connected to the system or is not powered on.
OPOS_E_OFFLINE	The device is off-line.
OPOS_E_NOEXIST	The file name (or other specified value) does not exist.
OPOS_E_EXISTS	The file name (or other specified value) already exists.
OPOS_E_FAILURE	The device cannot perform the requested procedure, even though the device is connected to the system, powered on, and on-line.
OPOS_E_TIMEOUT	The Service Object timed out waiting for a response from the device, or the Control timed out waiting for a response from the Service Object.
OPOS_E_BUSY	The current Service Object state does not allow this request. For example, if asynchronous output is in progress, certain methods may not be allowed.
OPOS_E_EXTENDED	A class-specific error condition occurred. The error condition code is available in the ResultCodeExtended property.

ResultCodeExtended Property

Syntax **LONG ResultCodeExtended;**

Remarks When the **ResultCode** is set to OPOS_E_EXTENDED, this property is set to a class-specific value, and must match one of the values given in this document under the appropriate device class section.

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

ServiceObjectDescription Property

Syntax **BSTR ServiceObjectDescription;**

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

A sample returned string is:

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

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

ServiceObjectVersion Property

Syntax **LONG ServiceObjectVersion;**

Remarks Service object version number.

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

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

A sample version number is:

```
1002038
```


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

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

Note:

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

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

State Property

Syntax **LONG State;**

Remarks Contains the current state of the Control.

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

This property is always readable.

Methods

CheckHealth Method

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

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

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

Remarks Called to test the state of a device.

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

The **CheckHealth** method is always synchronous.

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

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

ClaimDevice Method**Added in Release 1.5**

Syntax	<p>LONG ClaimDevice (LONG <i>Timeout</i>);</p> <p>The <i>Timeout</i> parameter gives the maximum number of milliseconds to wait for exclusive access to be satisfied.</p> <p>If zero, the method attempts to claim the device, then returns the appropriate status immediately.</p> <p>If OPOS_FOREVER (-1), the method waits as long as needed until exclusive access is satisfied.</p>								
Remarks	<p>Call this method to request 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> <p>Release 1.0 – 1.4 In releases prior to 1.5, this method is named Claim.</p> <p>Release 1.5 and later</p> <p>ClaimDevice must be used by early-bound applications. For compatibility with late-bound applications, the Control Object's IDispatch interface supports both ClaimDevice and Claim. It is recommended that applications written to the 1.5 specification use ClaimDevice, not Claim.</p> <p>Early bound applications acquire Control Object calling details at development time, including Class IDs, Interface IDs, and method, property, and event calling details. They then can build in static sequences to call methods and properties and receive events. Microsoft Visual C++ and Visual Basic plus most compiled languages support early binding.</p> <p>Late bound applications acquire calling details at run time. They then dynamically build code sequences to call methods and properties plus receive events. Scripting languages usually support late binding. Late binding can be implemented with many compiled languages, too, but often require additional programmer effort, especially to receive events.</p>								
Return	<p>One of the following values is returned by the method and also placed in the ResultCode property:</p> <table border="1"> <thead> <tr> <th style="text-align: left;">Value</th> <th style="text-align: left;">Meaning</th> </tr> </thead> <tbody> <tr> <td>OPOS_SUCCESS</td> <td>Exclusive access has been granted. The Claimed property is now TRUE. Also returned if this application has already claimed the device.</td> </tr> <tr> <td>OPOS_E_ILLEGAL</td> <td>This device cannot be claimed for exclusive access, or an invalid <i>Timeout</i> parameter was specified.</td> </tr> <tr> <td>OPOS_E_TIMEOUT</td> <td>Another application has exclusive access to the device, and did not relinquish control before <i>Timeout</i> milliseconds expired.</td> </tr> </tbody> </table>	Value	Meaning	OPOS_SUCCESS	Exclusive access has been granted. The Claimed property is now TRUE. Also returned if this application has already claimed the device.	OPOS_E_ILLEGAL	This device cannot be claimed for exclusive access, or an invalid <i>Timeout</i> parameter was specified.	OPOS_E_TIMEOUT	Another application has exclusive access to the device, and did not relinquish control before <i>Timeout</i> milliseconds expired.
Value	Meaning								
OPOS_SUCCESS	Exclusive access has been granted. The Claimed property is now TRUE. Also returned if this application has already claimed the device.								
OPOS_E_ILLEGAL	This device cannot be claimed for exclusive access, or an invalid <i>Timeout</i> parameter was specified.								
OPOS_E_TIMEOUT	Another application has exclusive access to the device, and did not relinquish control before <i>Timeout</i> milliseconds expired.								

ClearInput Method

Syntax **LONG ClearInput ();**

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

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

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

Value	Meaning
OPOS_SUCCESS	Input has been cleared.
OPOS_E_CLAIMED	The device is claimed by another process.
OPOS_E_NOTCLAIMED	The device must be claimed before this method can be used.

ClearOutput Method

Updated in Release 1.7

Syntax **LONG ClearOutput ();**

Remarks Called to clear all buffered output data, including all asynchronous output. Also, when possible, halts outputs that are in progress.

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

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

Value	Meaning
OPOS_SUCCESS	Output has been cleared.
OPOS_E_CLAIMED	The device is claimed by another process.
OPOS_E_NOTCLAIMED	The device must be claimed before this method can be used.

Close Method

Syntax **LONG Close ();**

Remarks Called to release the device and its resources.

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

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

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

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

DirectIO Method

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

Parameter	Description
<i>Command</i>	Command number. Specific values assigned by the Service Object.
<i>pData</i>	Pointer to additional numeric data. Specific values vary by <i>Command</i> and Service Object.
<i>pString</i>	Pointer to additional string data. Specific values vary by <i>Command</i> and Service Object. The format of this data depends upon the value of the BinaryConversion property. See page A-28.

Remarks Call to communicate directly with the Service Object.

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

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

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

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

Open Method

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

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

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

The device name specifies which of one or more devices supported by this Control Object should be used. The *DeviceName* must exist in the system registry for this device class. The relationship between the device name and physical devices is determined by entries within the operating system registry; these entries are maintained by a setup or configuration utility.

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

Release 1.5 and later

The value of the **OpenResult** property is set by the **Open** method.

Return One of the following values is returned by the method:

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

Note:

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

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

ReleaseDevice Method***Added in Release 1.5*****Syntax** **LONG ReleaseDevice ();****Remarks** Call this method to release exclusive access to the device.

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

Release 1.0 – 1.4

In releases prior to 1.5, this method is named **Release**.

Release 1.5 and later

ReleaseDevice must be used by early-bound applications. For compatibility with late-bound applications, the Control Object's IDispatch interface supports both **ReleaseDevice** and **Release**. It is recommended that applications written to the 1.5 specification use **ReleaseDevice**, not **Release**.

Early bound applications acquire Control Object calling details at development time, including Class IDs, Interface IDs, and method, property, and event calling details. They then can build in static sequences to call methods and properties and receive events. Microsoft Visual C++ and Visual Basic plus most compiled languages support early binding.

Late bound applications acquire calling details at run time. They then dynamically build code sequences to call methods and properties plus receive events. Scripting languages usually support late binding. Late binding can be implemented with many compiled languages, too, but often require additional programmer effort, especially to receive events.

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

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

ResetStatistics Method

Added in Release 1.8

Syntax **LONG ResetStatistics (BSTR StatisticsBuffer);**

Parameter	Description
<i>StatisticsBuffer</i>	The data buffer defining the statistics that are to be reset.

This is a comma-separated list of name(s), where an empty string (“”) means ALL resettable statistics are to be reset, “U_” means all UnifiedPOS defined resettable statistics are to be reset, “M_” means all manufacturer defined resettable statistics are to be reset, and “actual_name1, actual_name2” (from the XML file definitions) means that the specifically defined resettable statistic(s) are to be reset.

Remarks Resets the defined resettable statistics in a device.

Both **CapStatisticsReporting** and **CapUpdateStatistics** must be TRUE in order to successfully use this method.

This method is always executed synchronously.

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

Value	Meaning
OPOS_SUCCESS	The statistics have been reset.
OPOS_E_ILLEGAL	CapStatisticsReporting or CapUpdateStatistics is FALSE, or the named statistic is not defined/resettable.
<i>Other Values</i>	See ResultCode .

See Also **CapStatisticsReporting** Property, **CapUpdateStatistics** Property.

RetrieveStatistics Method

Added in Release 1.8

Syntax **LONG RetrieveStatistics (BSTR* pStatisticsBuffer);**

Parameter	Description
<i>statisticsBuffer</i>	The data buffer defining the statistics to be retrieved and in which the retrieved statistics are placed.

This is a comma-separated list of name(s), where an empty string (“”) means ALL statistics are to be retrieved, “U_” means all UnifiedPOS defined statistics are to be retrieved, “M_” means all manufacturer defined statistics are to be retrieved, and “actual_name1, actual_name2” (from the XML file definitions) means that the specifically defined statistic(s) are to be retrieved.

Remarks Retrieves the statistics from a device.

CapStatisticsReporting must be TRUE in order to successfully use this method.

This method is always executed synchronously.

All calls to **RetrieveStatistics** will return the following XML as a minimum:


```

<?xml version='1.0'?>
<UPOSStat version="1.8.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://www.nrf-arts.org/IXRetail/namespace/"
  xsi:schemaLocation=
    "http://www.nrf-arts.org/IXRetail/namespace/
    UPOSStat.xsd">
  <Event>
    <Parameter>
      <Name>RequestedStatistic</Name>
      <Value>1234</Value>
    </Parameter>
  </Event>
  <Equipment>
    <Manufacturer>
      <Name>Device Manufacturer</Name>
    </Manufacturer>
    <ModelNumber>Device Model Number</ModelNumber>
    <SensorID UPOS="POSPrinter"/>
  </Equipment>
</UPOSStat>

```

If the application requests a statistic name that the device does not support, the `<Parameter>` entry will be returned with an empty `<Value>`. e.g.,

```

<Parameter>
  <Name>RequestedStatistic</Name>
  <Value></Value>
</Parameter>

```

All statistics that the device collects that are manufacturer specific (not defined in the schema) will be returned in a `<ManufacturerSpecific>` tag instead of a `<Parameter>` tag. e.g.,

```

<ManufacturerSpecific>
  <Name>TheAnswer</Name>
  <Value>42</Value>
</ManufacturerSpecific>

```

When an application requests all statistics from the device, the device will return a `<Parameter>` entry for every defined statistic for the device category as defined by the XML schema version specified by the version attribute in the `<UPOSStat>` tag. If the device does not record any of the statistics, the `<Value>` tag will be empty.

[The most up-to-date files defining the XML tag names and example schemas for the statistics for all device categories can be downloaded from the NRF-ARTS web site at http://www.nrf-arts.org.](http://www.nrf-arts.org)

Return	One of the following values is returned by the method and also placed in the ResultCode property:								
	<table border="0" style="width: 100%;"> <thead> <tr> <th style="text-align: left;">Value</th> <th style="text-align: left;">Meaning</th> </tr> </thead> <tbody> <tr> <td>OPOS_SUCCESS</td> <td>The statistics have been retrieved and placed into the supplied buffer.</td> </tr> <tr> <td>OPOS_E_ILLEGAL</td> <td>CapStatisticsReporting is FALSE or the named statistic is not defined.</td> </tr> <tr> <td><i>Other Values</i></td> <td>See ResultCode.</td> </tr> </tbody> </table>	Value	Meaning	OPOS_SUCCESS	The statistics have been retrieved and placed into the supplied buffer.	OPOS_E_ILLEGAL	CapStatisticsReporting is FALSE or the named statistic is not defined.	<i>Other Values</i>	See ResultCode .
Value	Meaning								
OPOS_SUCCESS	The statistics have been retrieved and placed into the supplied buffer.								
OPOS_E_ILLEGAL	CapStatisticsReporting is FALSE or the named statistic is not defined.								
<i>Other Values</i>	See ResultCode .								
See Also	CapStatisticsReporting Property.								

UpdateStatistics Method

Added in Release 1.8

Syntax **LONG UpdateStatistics (BSTR *StatisticsBuffer*);**

Parameter	Description
<i>statisticsBuffer</i>	The data buffer defining the statistics with values that are to be updated.

This is a comma-separated list of name-value pair(s), where an empty string name (“”=value1”) means ALL resettable statistics are to be set to the value “value1”, “U_=value2” means all UnifiedPOS defined resettable statistics are to be set to the value “value2”, “M_=value3” means all manufacturer defined resettable statistics are to be set to the value “value3”, and “actual_name1=value4, actual_name2=value5” (from the XML file definitions) means that the specifically defined resettable statistic(s) are to be set to the specified value(s).

Remarks Updates the defined resettable statistics in a device.

Both **CapStatisticsReporting** and **CapUpdateStatistics** must be TRUE in order to successfully use this method.

This method is always executed synchronously.

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

Value	Meaning
OPOS_SUCCESS	The statistics have been reset.
OPOS_E_ILLEGAL	CapStatisticsReporting or CapUpdateStatistics is FALSE, or the named statistic is not defined/updatable.
<i>Other Values</i>	See ResultCode .

See Also **CapStatisticsReporting** Property, **CapUpdateStatistics** Property.

Events

DataEvent Event

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

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

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

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

DirectIOEvent Event

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

Parameter	Description
<i>EventNumber</i>	Event number. Specific values are assigned by the Service Object.
<i>pData</i>	Pointer to additional numeric data. Specific values vary by <i>EventNumber</i> and the Service Object.
<i>pString</i>	Pointer to additional string data. Specific values vary by <i>EventNumber</i> and the Service Object. The format of this data depends upon the value of the BinaryConversion property. See page A-28.

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

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

ErrorEvent Event**Updated in Release 1.7**

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

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

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

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

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

Value	Meaning
OPOS_ER_RETRY	Typically valid only when locus is OPOS_EL_OUTPUT. Retry the asynchronous output. The error state is exited. May be valid when locus is OPOS_EL_INPUT. Default when locus is OPOS_EL_OUTPUT.
OPOS_ER_CLEAR	Clear all buffered output data (including all asynchronous output) or buffered input data. The error state is exited. Default when locus is OPOS_EL_INPUT.
OPOS_ER_CONTINUEINPUT	Use only when locus is OPOS_EL_INPUT_DATA. Acknowledges the error and directs the Control to continue processing. The Control remains in the error

state and will deliver additional **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 OPOS_EL_INPUT.
Default when locus is OPOS_EL_INPUT_DATA.

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

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

OutputCompleteEvent Event

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

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

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

StatusUpdateEvent Event

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

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

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

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

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

Release 1.3 and later – Power State Reporting

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

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

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

Peripheral Interfaces

Note:

The following are two examples that attempt to show how a Visual Basic program and a VC++ program would use the commands in a typical MFC implementation. Where possible the tables are arranged to show the sequence of the commands for proper operation of the peripheral device.

The Cash Drawer and the MICR devices were chosen because they represent a simple output device and a more complex input device. The other peripheral devices would follow similar command usage and flow.

OPOS: Cash Drawer

Visual Basic Command Examples.

OPERATION	T Y P E	VISUAL BASIC SAMPLE	R E A D	W R I T E	A R G S	R T N V	R C	R C E	Ref Page
-----------	------------------	---------------------	------------------	-----------------------	------------------	------------------	--------	-------------	-------------

Initializing Properties, Methods, and Events

Open *	M	IResult = CashDrawer.Open("Standard")	•	•	1	LONG	•	•	52
ClaimDevice *	M	IResult = CashDrawer.ClaimDevice("1000")	•	•	1	LONG	•	•	A-45
Claimed	P	bResult = CashDrawer.Claimed	•			BOOL			40
DeviceEnabled *	P	CashDrawer.DeviceEnabled = True		•	1	-			42
DeviceEnabled	P	bResult = CashDrawer.DeviceEnabled	•			BOOL	•	•	42
DirectIO	M	IResult= CashDrawer.DirectIO(0,Ival,"[!"])	•	•	3	LONG	•	•	51
CheckHealth	M	IResult = CashDrawer.CheckHealth(OPOS_CH_INTERNAL)	•	•	1	LONG	•	•	49
DirectIOEvent	E	Private Sub CashDrawer_DirectIOEvent(ByVal EventNumber As Long, pData As Long, pString As String)			3	CMF			60

Capabilities, Assignments and Descriptions Properties, Methods, and Events

StatusUpdateEvent	E	Private Sub CashDrawer_StatusUpdateEvent(ByVal Status As Long)			1	CMF			63
BinaryConversion	P	CashDrawer.BinaryConversion = OPOS_BC_DECIMAL		•	1	-	•	•	A-28
BinaryConversion	P	IResult = CashDrawer.BinaryConversion	•			LONG			A-28
CapPowerReporting	P	IResult = CashDrawer.CapPowerReporting	•			LONG			38
CheckHealthText	P	sResult = CashDrawer.CheckHealthText	•			BSTR			38
FreezeEvents	P	CashDrawer.FreezeEvents = True		•	1	-	•	•	44
FreezeEvents	P	bResult = CashDrawer.FreezeEvents	•			BOOL			44
PowerNotify	P	CashDrawer.PowerNotify = OPOS_PN_ENABLED		•	1	-	•	•	45
PowerNotify	P	IResult = CashDrawer.PowerNotify	•			LONG			45
PowerState	P	IResult = CashDrawer.PowerState	•			LONG			46

OPERATION	T Y P E	VISUAL BASIC SAMPLE	R E A D	W R I T E	A R G S	R T N V	R C	R C E	Ref Page
ResultCode	P	IResult = CashDrawer.ResultCode	•			LONG			A-40
ResultCodeExtended	P	IResult = CashDrawer.ResultCodeExtended	•			LONG			A-42
State	P	IResult = CashDrawer.State	•			LONG			48
ControlObject Description	P	sResult = CashDrawer.ControlObjectDescription	•			BSTR			41
ControlObject Version	P	IResult = CashDrawer.ControlObjectVersion	•			LONG			41
ServiceObject Description	P	sResult = CashDrawer.ServiceObjectDescription	•			BSTR			42
ServiceObject Version	P	IResult = CashDrawer.ServiceObjectVersion	•			LONG			43
DeviceDescription	P	sResult = CashDrawer.DeviceDescription	•			BSTR			A-34
DeviceName	P	sResult = CashDrawer.DeviceName	•			BSTR			A-35

Cash Drawer Operations Properties and Methods

CapStatus	P	bResult = CashDrawer.CapStatus	•			BOOL			127
CapStatusMultiDrawerDetect	P	bResult = CashDrawer.CapStatusMultiDrawerDetect	•			BOOL			127
DrawerOpened	P	bResult = CashDrawer.DrawerOpened	•			BOOL			128
OpenDrawer *	M	IResult = CashDrawer.OpenDrawer	•	•		LONG	•	•	129
WaitForDrawerClose	M	IResult = CashDrawer.WaitForDrawerClose(2500, 1000, 10, 5)	•	•	4	LONG	•	•	129

Terminating Methods

ReleaseDevice	M	IResult = CashDrawer.ReleaseDevice	•	•		LONG	•	•	A-49
Close *	M	IResult = CashDrawer.Close	•	•		LONG	•	•	51

Notes:

* Required for basic Cash Drawer operations

Legends:

TYPE = (P)roperty, (M)ethod, or (E)vent

ARGS = Number of Arguments Expected

RTNV = Return Value

'CMF' = Class Member Function

RC = Result Code

RCE = Result Code Extended

Ref Page = Page Number of UnifiedPOS Reference Description

Visual C++ Command Examples.

OPERATION	T Y P E	VISUAL C++ SAMPLE	R E A D	W R I T E	A R G S	R T N V	R C	R C E	Ref Page
-----------	------------------	-------------------	------------------	-----------------------	------------------	------------------	--------	-------------	-------------

Initializing Properties, Methods, and Events

Open *	M	IResult = m_CashDrawer.Open("Standard");	•	•	1	LONG	•	•	52
ClaimDevice *	M	IResult = m_CashDrawer.ClaimDevice("1000");	•	•	1	LONG	•	•	A-45
Claimed	P	bResult = m_CashDrawer.GetClaimed();	•			BOOL			40
DeviceEnabled *	P	m_CashDrawer.SetDeviceEnabled(TRUE);		•	1	-			42
DeviceEnabled	P	bResult = m_CashDrawer.GetDeviceEnabled();	•			BOOL	•	•	42
DirectIO	M	IResult = m_CashDrawer.DirectIO(0,&lval, "[")	•	•	3	LONG	•	•	51
CheckHealth	M	IResult = m_CashDrawer.CheckHealth(OPOS_CH_INTERNAL);	•	•	1	LONG	•	•	49
DirectIOEvent	E	void COCashDrawerDlg::OnDirectIOEventCashDrawerctrl(long EventNumber, long FAR* pData, BSTR FAR* pString)			3	CMF			60

Capabilities, Assignments and Descriptions Properties, Methods, and Events

StatusUpdateEvent	E	void COCashDrawerDlg::OnStatusUpdateEventCashDrawerctrl(long Status)			1	CMF			63
BinaryConversion	P	m_CashDrawer.SetBinaryConversion(OPOS_BC_DECIMAL);		•	1	-	•	•	A-28
BinaryConversion	P	IResult = m_CashDrawer.GetBinaryConversion();	•			LONG			A-28
CapPowerReporting	P	IResult = m_CashDrawer.GetCapPowerReporting();	•			LONG			38
CheckHealthText	P	sResult = m_CashDrawer.GetCheckHealthText();	•			BSTR			38
FreezeEvents	P	m_CashDrawer.SetFreezeEvents(TRUE);		•	1	-	•	•	44
FreezeEvents	P	bResult = m_CashDrawer.GetFreezeEvents();	•			BOOL			44
PowerNotify	P	m_CashDrawer.SetPowerNotify(OPOS_PN_ENABLED);		•	1	-	•	•	45
PowerNotify	P	IResult = m_CashDrawer.GetPowerNotify();	•			LONG			45
PowerState	P	IResult = m_CashDrawer.GetPowerState();	•			LONG			46
ResultCode	P	IResult = m_CashDrawer.GetResultCode();	•			LONG			A-40
ResultCodeExtended	P	IResult = m_CashDrawer.GetResultCodeExtended();	•			LONG			A-42

OPERATION	T Y P E	VISUAL C++ SAMPLE	R E A D	W R I T E	A R G S	R T N V	R C	R C E	Ref Page
State	P	IResult = m_CashDrawer.GetState ();	•			LONG			48
ControlObject Description	P	sResult = m_CashDrawer.GetControlObjectDescription();	•			BSTR			41
ControlObject Version	P	IResult = m_CashDrawer.GetControlObjectVersion();	•			LONG			41
ServiceObject Description	P	sResult = m_CashDrawer.GetServiceObjectDescription();	•			BSTR			42
ServiceObject Version	P	IResult = m_CashDrawer.GetServiceObjectVersion();	•			LONG			43
DeviceDescription	P	sResult = m_CashDrawer.GetDeviceDescription();	•			BSTR			47
DeviceName	P	sResult = m_CashDrawer.GetDeviceName();	•			BSTR			47

Cash Drawer Operations Properties and Methods

CapStatus	P	bResult = m_CashDrawer.GetCapStatus();	•			BOOL			127
CapStatusMultiDrawerDetect	P	bResult = m_CashDrawer.GetCapStatusMultiDrawerDetect();	•			BOOL			127
DrawerOpened	P	bResult = m_CashDrawer.GetDrawerOpened();	•			BOOL			128
OpenDrawer *	M	IResult = m_CashDrawer.OpenDrawer();	•	•		LONG	•	•	129
WaitForDrawerClose	M	IResult = m_CashDrawer.WaitForDrawerClose(2500, 1000, 10, 5);	•	•	4	LONG	•	•	129

Terminating Methods

ReleaseDevice	M	IResult = m_CashDrawer.ReleaseDevice();	•	•		LONG	•	•	A-49
Close *	M	IResult = m_CashDrawer.Close();	•	•		LONG	•	•	51

Notes:

* Required for basic Cash Drawer operations

Legends:

TYPE = (P)roperty, (M)ethod, or (E)vent
 ARGS = Number of Arguments Expected
 RTNV = Return Value
 'CMF' = Class Member Function
 RC = Result Code
 RCE = Result Code Extended
 Ref Page = Page Number of UnifiedPOS Reference Description

OPOS: MICR

Visual Basic Command Examples.

OPERATION	T Y P E	VISUAL BASIC SAMPLE	R E A D	W R I T E	A R G S	R T N V	R C	R C E	Ref Page
-----------	------------------	---------------------	------------------	-----------------------	------------------	------------------	--------	-------------	-------------

Initializing Properties, Methods, and Events

Open *	M	IResult = Micr.Open("M101")	•	•	1	LONG	•	•	52
ClaimDevice *	M	IResult = Micr.ClaimDevice("1000")	•	•	1	LONG	•	•	A-45
Claimed	P	bResult = Micr.Claimed	•			BOOL			40
DeviceEnabled *	P	Micr.DeviceEnabled = True		•	1	-	•	•	42
DeviceEnabled	P	bResult = Micr.DeviceEnabled	•			BOOL			42
AutoDisable	P	Micr.AutoDisable = True		•	1	-	•	•	38
AutoDisable	P	bResult = Micr.AutoDisable	•		1	BOOL			38
DirectIO	M	IResult= Micr.DirectIO(0,Ival,"0x1b")	•	•	3	LONG	•	•	51
CheckHealth	M	IResult = Micr.CheckHealth(OPOS_CH_INTERNAL)	•	•	1	LONG	•	•	49
DirectIOEvent	E	Private Sub Micr_DirectIOEvent(ByVal EventNumber As Long, pData As Long, pString As String)			3	CMF			60
ErrorEvent	E	Private Sub Micr_ErrorEvent(ByVal ResultCode As Long, ByVal ResultCodeExtended As Long, ByVal ErrorLocus As Long, pErrorResponse As Long)			4	CMF			61

Capabilities, Assignments and Descriptions Properties, Methods, and Events

StatusUpdateEvent	E	Private Sub Micr_StatusUpdateEvent(ByVal Status As Long)			1	CMF			63
BinaryConversion	P	Micr.BinaryConversion = OPOS_BC_DECIMAL		•	1	-	•	•	A-28
BinaryConversion	P	IResult = Micr.BinaryConversion	•			LONG			A-28
CapPowerReporting	P	IResult = Micr.CapPowerReporting	•			LONG			38
CheckHealthText	P	sResult = Micr.CheckHealthText	•			BSTR			38
DataCount	P	IResult = Micr.DataCount	•			LONG			40
FreezeEvents	P	Micr.FreezeEvents = True		•	1	-	•	•	44
FreezeEvents	P	bResult = Micr.FreezeEvents	•			BOOL			44

OPERATION	T Y P E	VISUAL BASIC SAMPLE	R E A D	W R I T E	A R G S	R T N V	R C	R C E	Ref Page
PowerNotify	P	Micr.PowerNotify = OPOS_PN_ENABLED		•	1	-	•	•	45
PowerNotify	P	IResult = Micr.PowerNotify	•			LONG			45
PowerState	P	IResult = Micr.PowerState	•			LONG			46
ResultCode	P	IResult = Micr.ResultCode	•			LONG			A-40
ResultCodeExtended	P	IResult = Micr.ResultCodeExtended	•			LONG			A-42
State	P	IResult = Micr.State	•			LONG			48
ControlObject Description	P	sResult = Micr.ControlObjectDescription	•			BSTR			41
ControlObject Version	P	IResult = Micr.ControlObjectVersion	•			LONG			41
ServiceObject Description	P	sResult = Micr.ServiceObjectDescription	•			BSTR			42
ServiceObject Version	P	IResult = Micr.ServiceObjectVersion	•			LONG			43
DeviceDescription	P	sResult = Micr.DeviceDescription	•			BSTR			47
DeviceName	P	sResult = Micr.DeviceName	•			BSTR			47

MICR Operations Properties, Methods, and Events

CapValidationDevice	P	bResult = Micr.CapValidationDevice	•			BOOL			461
ClearInput	M	IResult = Micr.ClearInput	•	•		LONG	•	•	50
DataEventEnabled *	P	Micr.DataEventEnabled = True		•	1	-	•	•	40
DataEventEnabled	P	bResult = Micr.DataEventEnabled	•			BOOL			40
BeginInsertion *	M	IResult = Micr.BeginInsertion	•	•		LONG	•	•	464
EndInsertion *	M	IResult = Micr.EndInsertion	•	•		LONG	•	•	466
DataEvent	E	Private Sub Micr_DataEvent(ByVal Status As Long)			1	CMF			59
BeginRemoval *	M	IResult = Micr.BeginRemoval	•	•		LONG	•	•	465
EndRemoval *	M	IResult = Micr.EndRemoval	•	•		LONG	•	•	467
RawData	P	sResult = Micr.RawData	•			BSTR			463
AccountNumber	P	sResult = Micr.AccountNumber	•			BSTR			460

OPERATION	T Y P E	VISUAL BASIC SAMPLE	R E A D	W R I T E	A R G S	R T N V	R C	R C E	Ref Page
Amount	P	sResult = Micr.Amount	•			BSTR			460
BankNumber	P	sResult = Micr.BankNumber	•			BSTR			460
EPC	P	sResult = Micr.EPC	•			BSTR			462
SerialNumber	P	sResult = Micr.SerialNumber	•			BSTR			463
TransitNumber	P	sResult = Micr.TransitNumber	•			BSTR			463
CheckType	P	IResult = Micr.CheckType	•			LONG			461
CountryCode	P	IResult = Micr.CountryCode	•			LONG			462

Terminating Methods

ReleaseDevice	M	IResult = Micr.ReleaseDevice	•	•		LONG	•	•	A-49
Close *	M	IResult = Micr.Close	•	•		LONG	•	•	51

Notes:

* Required for basic MICR operations

Legends:

TYPE = (P)roperty, (M)ethod, or (E)vent
 ARGS = Number of Arguments Expected
 RTNV = Return Value
 'CMF' = Class Member Function
 RC = Result Code
 RCE = Result Code Extended
 Ref Page = Page Number of UnifiedPOS Reference Description

Visual C++ Command Examples.

OPERATION	T Y P E	VISUAL C++ SAMPLE	R E A D	W R I T E	A R G S	R T N V	R C	R C E	Ref Page
-----------	------------------	-------------------	------------------	-----------------------	------------------	------------------	--------	-------------	-------------

Initializing Properties, Methods, and Events

Open *	M	IResult = m_Micr.Open("M101");	•	•	1	LONG	•	•	52
ClaimDevice *	M	IResult = m_Micr.ClaimDevice("1000");	•	•	1	LONG	•	•	A-45
Claimed	P	bResult = m_Micr.GetClaimed();	•			BOOL			40
DeviceEnabled *	P	m_Micr.SetDeviceEnabled(TRUE);		•	1	-	•	•	42
DeviceEnabled	P	bResult = m_Micr.GetDeviceEnabled();	•			BOOL			42
AutoDisable	P	m_Micr.SetAutoDisable(TRUE);		•	1	-	•	•	38
AutoDisable	P	bResult m_Micr.GetAutoDisable();	•		1	BOOL			38
DirectIO	M	IResult = m_Micr.DirectIO(0,&lval,"0x1b")	•	•	3	LONG	•	•	51
CheckHealth	M	IResult = m_Micr.CheckHealth(OPOS_CH_INTERNAL);	•	•	1	LONG	•	•	49
DirectIOEvent	E	void COMicrDlg::OnDirectIOEventMicrctrl(long EventNumber, long FAR* pData, BSTR FAR* pString)			3	CMF			60
ErrorEvent	E	void COMicrDlg::OnErrorEventMicrctrl(long ResultCode, long ResultCodeExtended, long ErrorLocus, long FAR* pErrorResponse)			4	CMF			61

Capabilities, Assignments and Descriptions Properties, Methods, and Events

StatusUpdateEvent	E	void COMicrDlg::OnStatusUpdateEventMicrctrl (long Status)			1	CMF			63
BinaryConversion	P	m_Micr.SetBinaryConversion(OPOS_BC_DECIMAL);		•	1	-	•	•	A-28
BinaryConversion	P	IResult = m_Micr.GetBinaryConversion();	•			LONG			A-28
CapPowerReporting	P	IResult = m_Micr.GetCapPowerReporting();	•			LONG			38
CheckHealthText	P	sResult = m_Micr.GetCheckHealthText();	•			BSTR			38
DataCount	P	IResult = m_Micr.GetDataCount();	•			LONG			40
FreezeEvents	P	m_Micr.SetFreezeEvents(TRUE);		•	1	-	•	•	44
FreezeEvents	P	bResult = m_Micr.GetFreezeEvents();	•			BOOL			44

OPERATION	T Y P E	VISUAL C++ SAMPLE	R E A D	W R I T E	A R G S	R T N V	R C	R C E	Ref Page
PowerNotify	P	m_Micr.SetPowerNotify(OPOS_PN_ENABLED);		•	1	-	•	•	45
PowerNotify	P	IResult = m_Micr.GetPowerNotify();	•			LONG			45
PowerState	P	IResult = m_Micr.GetPowerState();	•			LONG			46
ResultCode	P	IResult = m_Micr.GetResultCode();	•			LONG			A-40
ResultCodeExtended	P	IResult = m_Micr.GetResultCodeExtended();	•			LONG			A-42
State	P	IResult = m_Micr.GetState();	•			LONG			48
ControlObject Description	P	sResult = m_Micr.GetControlObjectDescription();	•			BSTR			41
ControlObject Version	P	IResult = m_Micr.GetControlObjectVersion();	•			LONG			41
ServiceObject Description	P	sResult = m_Micr.GetServiceObjectDescription();	•			BSTR			42
ServiceObject Version	P	IResult = m_Micr.GetServiceObjectVersion();	•			LONG			43
DeviceDescription	P	sResult = m_Micr.GetDeviceDescription();	•			BSTR			47
DeviceName	P	sResult = m_Micr.GetDeviceName();	•			BSTR			47

MICR Operations Properties, Methods, and Events

CapValidationDevice	P	bResult = m_Micr.GetCapValidationDevice();	•			BOOL			461
ClearInput	M	IResult = m_Micr.ClearInput();	•	•		LONG	•	•	50
DataEventEnabled *	P	m_Micr.SetDataEventEnabled(TRUE);		•	1	-	•	•	40
DataEventEnabled	P	bResult = m_Micr.GetDataEventEnabled();	•			BOOL			40
BeginInsertion *	M	IResult = m_Micr.BeginInsertion();	•	•		LONG	•	•	464
EndInsertion *	M	IResult = m_Micr.EndInsertion();	•	•		LONG	•	•	466
DataEvent	E	void COMicrDlg::OnDirectIOEventMicrtrl(long Status)			1	CMF			60
BeginRemoval *	M	IResult = m_Micr.BeginRemoval();	•	•		LONG	•	•	465
EndRemoval *	M	IResult = m_Micr.EndRemoval();	•	•		LONG	•	•	467
RawData	P	sResult = m_Micr.GetRawData();	•			BSTR			463
AccountNumber	P	sResult = m_Micr.GetAccountNumber();	•			BSTR			460

OPERATION	T Y P E	VISUAL C++ SAMPLE	R E A D	W R I T E	A R G S	R T N V	R C	R C E	Ref Page
Amount	P	sResult = m_Micr.GetAmount();	•			BSTR			460
BankNumber	P	sResult = m_Micr.GetBankNumber();	•			BSTR			460
EPC	P	sResult = m_Micr.GetEPC();	•			BSTR			462
SerialNumber	P	sResult = m_Micr.GetSerialNumber();	•			BSTR			463
TransitNumber	P	sResult = m_Micr.GetTransitNumber();	•			BSTR			463
CheckType	P	IResult = m_Micr.GetCheckType();	•			LONG			461
CountryCode	P	IResult = m_Micr.GetCountryCode();	•			LONG			462

Terminating Methods

ReleaseDevice	M	IResult = m_Micr.ReleaseDevice();	•	•		LONG	•	•	A-49
Close *	M	IResult = m_Micr.Close();	•	•		LONG	•	•	51

Notes:

* Required for basic MICR operations

Legends:

TYPE = (P)roperty, (M)ethod, or (E)vent
 ARGS = Number of Arguments Expected
 RTNV = Return Value
 'CMF' = Class Member Function
 RC = Result Code
 RCE = Result Code Extended
 Ref Page = Page Number of UnifiedPOS Reference Description

Section 3: OPOS Registry Usage

OPOS Controls require some data in the system registry in order for the Control Objects to locate the proper Service Object and initialize it for the device.

The registry is organized in a hierarchical structure, in which each level is named a “key.” Each key may contain:

- Additional keys (sometimes called “subkeys”).
- Zero or more named “values.” A value is assigned “data” of type string, binary, or double-word.
- One “default value” that may be assigned data of type string.

OPOS only defines string data.

Service Object Root Registry Key

All OPOS Service Object entries should be placed under the following main key:

HKEY_LOCAL_MACHINE\SOFTWARE\OLEforRetail\ServiceOPOS

The “HKEY_LOCAL_MACHINE\SOFTWARE” key is the recommended key for software configuration local to the PC. The “OLEforRetail” key will group all OLE for Retail related configuration information. The “ServiceOPOS” key maintains configuration information for OPOS Service Objects.

Device Class Keys

Each class has an identifying Device Class subkey under the main OPOS key. The following key names have been established:

BumpBar
CashChanger
CashDrawer
CAT
CoinDispenser
CheckScanner
FiscalPrinter
HardTotals
Keylock
LineDisplay
MICR
MotionSensor
MSR
PINPad
PointCardRW
POSKeyboard
POSPower
POSPrinter
RemoteOrderDisplay
Scale
Scanner
SignatureCapture
SmartCardRW
ToneIndicator

Device Name Keys and Values

Each device within a class is assigned a Device Name subkey under the class's key. This should be performed by a Service Object installation procedure. This Device Name key is passed to the Control Object's **Open** method by the application. The Device Name is not constrained, except that it must be unique among the names under the device class.

The default value of the Device Name key is the programmatic ID¹¹ of the Service Object. This string is needed by the Control Object, so that the Service Object may be loaded and the OLE Automation interfaces established between the CO and the SO.

<i>Value – Required</i>	<i>Data</i>
(Default)	Service Object's OLE Programmatic ID.

The device unit key's values and their data describe the characteristics of the actual device on the terminal or PC. The following values are strongly recommended for use by installation and support personnel:

<i>Value – Recommended</i>	<i>Data</i>
Service	Filename of the Service Object.
Description	String describing the Service Object.
Version	String containing the Service Object version number. General format is: MajorVersion.MinorVersion.BuildVersion.

Other values may be defined as needed by the Service Object. Values might contain information such as:

- Communications Port
- Baud Rate
- Serial Line Characteristics
- Interrupt Request (IRQ) Values
- Input/Output (I/O) Ports

Logical Device Name Values

An application may open a Control by passing the Device Name key to the **Open** method. In many cases, however, the application will want a level of isolation where the application specifies a "Logical Device Name" that is translated into a Device Name.

A Logical Device Name is added to the registry as a value contained in the Device Class key. The value name is set to the Logical Device Name, and its data must match a Device Name key contained in the same Device Class.

The application integrator is responsible for adding Logical Device Names to the registry. (They are not added by the Service Object install procedure.)

Service Provider Root Registry Key

The SO service providers may need to store some information in the registry that is common to some or all of its Service Objects. This data could include installation directories, installation date, and de-install information. Service provider information should be placed under the following main key:

HKEY_LOCAL_MACHINE\SOFTWARE\OLEforRetail\ServiceInfo

The subkeys under this key should be the names of service provider companies. Subkeys and values within each service provider company subkey are provider-dependent.

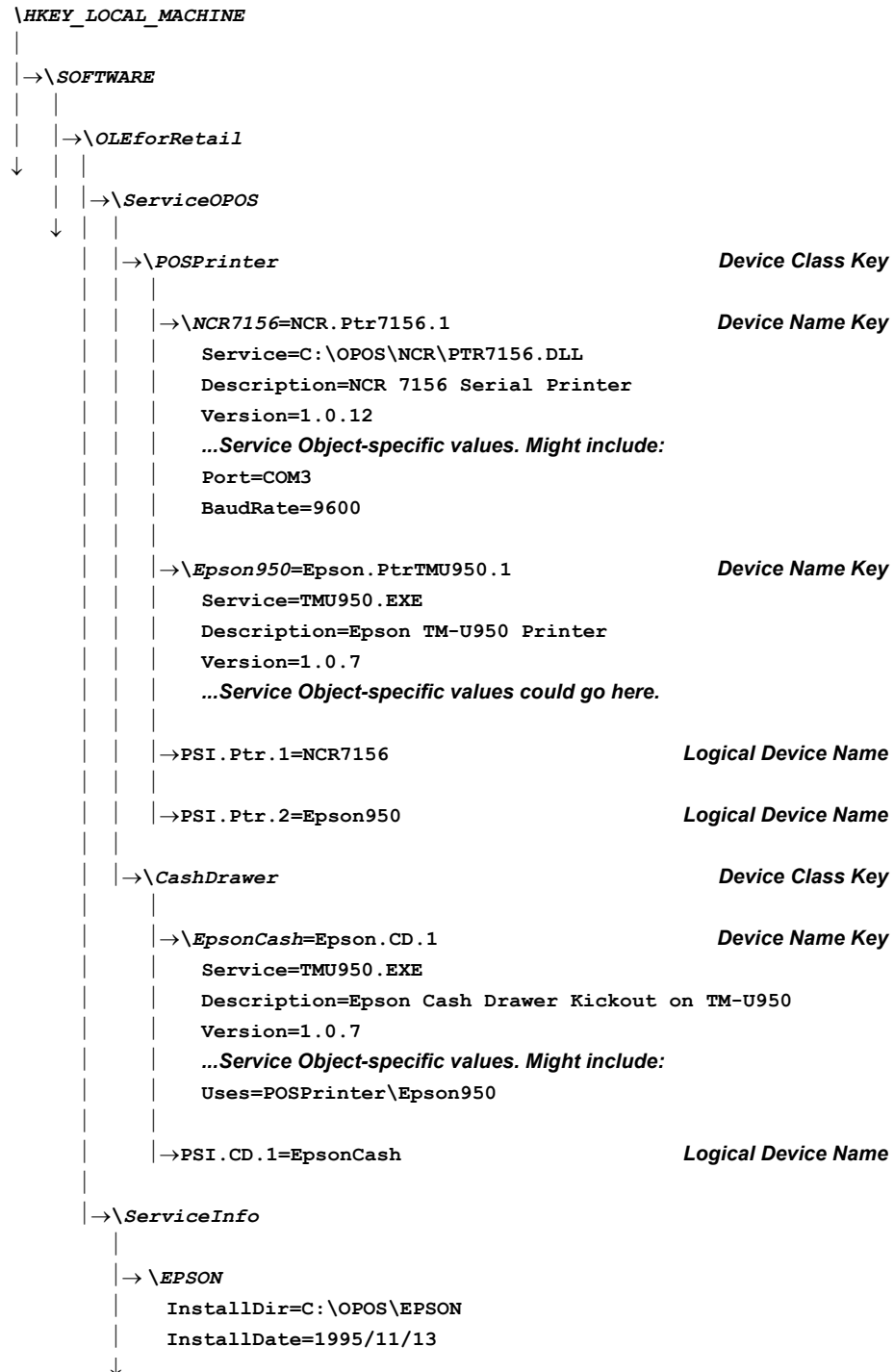
Example

In this example, keys are listed in *italics*. Comments appear as **comment**.

Two device classes are given: POSPrinter and CashDrawer.

The POSPrinter class contains two Device Names. Also, two Logical Device Names are present, which point to the Device Names.

The CashDrawer class contains one Device Name and one Logical Device Name. The Service Object has a unique ProgID but uses the same executable as one of the printers. This Service Object could use the example value “*uses*” to point to some registry values of the printer device that can be used for the cash drawer parameters.



Section 4: OPOS Application Header Files

The header files are listed in alphabetical order. The mapping of device class name to header file name is as follows:

General	Opos.h
Bump Bar	OposBb.h
Cash Changer	OposChan.h
Cash Drawer	OposCash.h
CAT	OposCat.h
Check Scanner	OposChk.h
Coin Dispenser	OposCoin.h
Fiscal Printer	OposFptr.h
Hard Totals	OposTot.h
Keylock	OposLock.h
Line Display	OposDisp.h
MICR	OposMicr.h
Motion Sensor	OposMotion.h
MSR	OposMsr.h
PIN Pad	OposPpad.h
Point Card Reader Writer	OposPerw.h
POS Keyboard	OposKbd.h
POS Power	OposPwr.h
POS Printer	OposPtr.h
Remote Order Display	OposRod.h
Scale	OposScal.h
Scanner	OposScan.h
Signature Capture	OposSig.h
Smart Card Reader Writer	OposScr.w
Tone Indicator	OposTone.h

The most up to date header files can be downloaded from the web site,

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

under the OPOS standard files section.

Section 5: Technical Details

System Strings (BSTR)

System String Characteristics

OPOS uses OLE system strings to pass and return data of variable length. System strings are often referred to as BStrings, and are assigned the type BSTR by Microsoft Visual C++.

A system string consists of a sequence of Unicode characters, which are each 16-bits wide. Thus, they are also referred to as “wide” characters. The string is followed by a NUL, or zero, character. The string is preceded by an unsigned long count of the bytes in the string, not including the NUL. Divide this count by two to obtain the number of characters in the string.

Most of the time, OPOS uses system strings to pass character data back and forth among the Application, Control Object, and System Object. A system string (BSTR) is used to pass string parameters by methods and to return string properties. A pointer to a system string (BSTR*) is used as a method parameter when the method must return string data.

System String Usage

Visual Basic both receives and sends system strings without any complications. The internal representation of VB strings is as wide characters with a length component. A BSTR may be passed using a variable, a string expression, or a literal. A BSTR* requires use of a variable, so that the data may be modified by the method.

Similarly, Visual C++ using ATL is straightforward. BSTR and BSTR* data is passed and received using these types. Any translation to or from Unicode is the developer’s responsibility.

Visual C++ with MFC, however, requires more consideration.

BSTR is handled as follows:

- BSTR Method Parameters
 - **Calling Function:** Calling an automation method with a BSTR parameter is treated by MFC as a pointer to a character string, LPCTSTR. If the VC++ ANSI option is used, MFC automatically converts from ANSI to Unicode.
 - **Called Function:** The function implementing an automation method receives a BSTR parameter as a pointer to a character string, LPCTSTR. If the VC++ ANSI option is used, then MFC performs an automatic conversion from Unicode into ANSI before passing control to the function. The string length immediately precedes the string pointer.
- BSTR Return Type (used for getting properties)
 - **Calling Function:** An automation method returning a BSTR result is automatically converted by MFC into a CString.

- **Called Function:** An automation method returns a BSTR result by placing the data into an MFC CString object, and returning the result of the CString's "AllocSysString" member function. If the VC++ ANSI option is used, then this function automatically converts the string from ANSI into Unicode.

BSTR* is passed and received by MFC as BSTR*, so the developer handling is the same as with ATL. Some MFC macros and classes may be helpful:

- If the VC++ ANSI option is used, then conversion between Unicode and MBCS is required. Some macros are available that make this conversion easier, such as T2OLE and OLE2T. (These do not handle NUL characters embedded in the string, however.)
- To set the string, place the data into an MFC CString object, and use CString's "SetSysString" member function.

System Strings and Binary Data

Sometimes OPOS uses BSTR and BSTR* to pass binary data.

These cases may return byte data in the range 00-hex to FF-hex. Each 16-bit character of the system string contains one byte of binary data in the lower 8 bits. The upper 8 bits are zero. This can lead to two problematic areas:

- The NUL character, or zero. Although system strings have a length component, some software still relies upon the NUL character to determine the end of the string.
- Characters in the range 0x80 – 0xFF. The translation between ANSI and Unicode formats may yield incorrect data, especially for eastern languages.

In order to avoid these translation and transmission problems, an Application should employ the **BinaryConversion** feature if data outside the range of 0x01 – 0x7F may be sent or received by a method parameter or a property.

BinaryConversion, added in Release 1.2, supports two means of converting data between binary and ASCII formats.

Mapping of CharSet *Added in Release 1.7*

For some devices like POSPrinter, LineDisplay, ROD, and PCRW, it is necessary to select an appropriate character set in the **CharSet** property of the Service when printing or displaying characters on the device. Usually an OPOS application uses the character set which is set by the operating system. This is, for example, the code page 1252 for the US and West-European countries and, for example, 1250 for East-European countries. The selected code page is the so-called ANSI code page. When the device only contains three-digit code pages - such as 850 or 852 - a mapping of the characters from their positions in the application-side code page to the device-side code page is necessary.

The following code snippet allows Service Object providers to easily add the mapping mechanism into their Services. (It is assumed that the data transferred to the Service for output to the device is already transformed from BSTR to LPCTSTR.)

```

BOOL AnsiToOEMCodePage(
    UINT CodePage,          // the desired destination code page like 858
    LPCTSTR src,           // source string assumed to be ACP (default
                          // system code page)
    INT srcLength,         // the length of the source string
    LPTSTR dest,           // destination String; when called 'dest'
                          // shows to a reserved area of 'destLength'-
    INT *destLength)       // bytes length of the destination string
{
    LPWSTR lpWideCharStr = NULL;
    INT WideCharStrLen = (srcLength+1)* sizeof(lpWideCharStr[0]);
    lpWideCharStr = (LPWSTR) malloc (WideCharStrLen);

    if (lpWideCharStr == NULL)
        return FALSE;

    // convert to UNICODE
    WideCharStrLen = MultiByteToWideChar (CP_ACP, 0, src, srcLength,
                                          lpWideCharStr, WideCharStrLen);

    if (WideCharStrLen<=0)
    {
        free (lpWideCharStr);
        return FALSE;
    }
    // convert UNICODE back to desired codepage ;
    // non mappable characters are mapped to space character
    const char defaultChar = 0x20;
    *destLength = WideCharToMultiByte (CodePage, 0, lpWideCharStr,
                                       WideCharStrLen, dest, *destLength, &defaultChar, NULL);
    free (lpWideCharStr);
    if (*destLength == 0 && WideCharStrLen != 0)// cp does not exist
        return FALSE;
    return TRUE;
}

```

Note:

- The code page currently selected in the system can be found in the Registry under: **HKLM\System\CurrentControlSet\Control\Nls\Codepage\ACP**.
- The destination code page must of course be installed when using the system API calls for mapping.

Section 6: Release 1.5 API Change: ClaimDevice and ReleaseDevice

The common methods **Claim** and **Release** were defined in the very first OPOS release. Since that time, an increased number of conflicts have occurred between the OPOS **Release** method and the COM **Release** method, which is a required method of every COM object. This conflict has required some development restrictions:

- Control Objects and Service Objects must define their interfaces as pure dispatch interfaces. This has precluded the use of the Microsoft Visual C++ Active Template Library, since ATL only supports IDispatch via a dual interface implementation.
- Some development environments assume that ActiveX Controls will not define a dispatch method that conflicts with COM. For example, users of Delphi have had to work around the **Release** conflict. Future tools may be even less tolerant of this conflict.

Therefore, these methods have been renamed to **ClaimDevice** and **ReleaseDevice** in Release 1.5.

Several steps have been taken to provide a maximal migration of Applications and Service Objects. These have been implemented in the reference set of Control Objects known as the “Common Control Objects”:

- Application.
Both the **ClaimDevice** and **Claim** methods and the **ReleaseDevice** and **Release** methods are supported by the Control Object’s IDispatch interface. The IDispatch interface is used by an application to implement late binding. By doing this, full backward compatibility is provided for current late bound Applications.

If an application using a development environment that performs early binding (including Microsoft Visual C++ and Visual Basic) changes from a 1.4 or earlier Control Object to a 1.5 or later Control Object, then it will also have to update all **Claim** calls to **ClaimDevice**, and **Release** calls to **ReleaseDevice**.

- Service Object.
A Service Object may expose either the **Claim** or **ClaimDevice** method and either the **Release** or **ReleaseDevice** method through its IDispatch interface. Note that if the Service Object is implemented using ATL, then it must use **ReleaseDevice**, since **Release** is reserved for COM’s IUnknown reference counting.

When the Application calls **ClaimDevice** or **Claim**, the Control Object calls the Service Object method **ClaimDevice** if present; otherwise it calls **Claim**. When the Application calls **ReleaseDevice** or **Release**, the Control Object calls the Service Object method **ReleaseDevice** if present; otherwise it calls **Release**. By doing this, full backward compatibility is provided for current Service Objects while allowing new Service Objects to be implemented using ATL.

Section 7: OPOS APG Change History

Release 1.01

Release 1.01 mostly adds clarifications and corrections, but the Line Display and Signature Capture chapters received substantive changes to correct deficiencies in their definition.

Release 1.01 replaces Release 1.0. The **ControlObjectVersion** for a compliant Control Object is 1000xxx, where xxx is a vendor-specific build number. The **ServiceObjectVersion** for a compliant Service Object is 1000xxx, where xxx is a vendor-specific build number.

Section	Change
Second Page	Add name of Microsoft Web site for OPOS information.
Introduction When ... Properties May Be Accessed	Update to say that capabilities are initialized at Open , others may not be initialized until DeviceEnabled = TRUE , and properties remain initialized until the Control is closed.
Introduction Device Sharing Model	If an exclusive device is Released , then reClaimed , settable device characteristics are restored to their state at Release .
Common Release method	If device is enabled, then disable before releasing.
Cash Drawer WaitForDrawerClose method	<i>BeepFrequency</i> is in hertz.
Hard Totals General Information	Recommend claiming necessary files before a BeginTrans , to ensure that CommitTrans does not fail.
Keylock General Information	Claim will return OPOS_E_ILLEGAL, not success.
Line Display General Information	Major clarification of line display usage modes; including intercharacter wait and marquee.
Line Display MarqueeFormat property	Add this property.
Line Display MarqueeType property	Add DISP_MT_INIT value.
Line Display ClearText and RefreshWindow methods	Clarify their functionality.
POS Printer XxxLetterQuality properties	Add initialization information.

- POS Printer **XxxLineWidth** properties
Clarify these properties.
- POS Printer **CapConcurrentXxxXxx** properties
Clarify that if a “concurrent” capability is false, then the application should print to only one of the stations at a time, and not alternate print lines between them.
- POS Printer **CapXxxNearendSensor** properties
Rename to **CapXxxNearEndSensor** for consistency with **XxxNearEnd** properties.
- POS Printer **CapXxxBarcode** properties
Rename to **CapXxxBarCode** for consistency with **PrintBarCode** method.
- Scale **Summary** Change **ClearInput** method to *Not Supported*. Scale input is not event-driven.
- Scale **WeightUnit** property
Change to read-only property.
- Signature Capture **MaximumX** and **MaximumY** properties
Clarify that maximum value is 65,535.
- Signature Capture **TotalVectors** and **VectorArray** properties
Rename to **TotalPoints** and **PointArray**. Update the **General Information** and the property remarks sections for consistency.
- Signature Capture **PointArray** property
Clarify that each point is represented by four characters: x (low 8 bits), x (high 8 bits), y (low 8 bits), y (high 8 bits).
- Throughout Update the property initialization details.
- OposDisp.h header file
Add **DISP_MT_INIT** constant and **MarqueeFormat** constants.
- Appendix C **Technical Details**
Add this appendix, with the sections:
- System strings and binary data.
- Event Handler Restrictions.

Release 1.1

Release 1.1 adds APIs based on requirements from OPOS-J, the Japanese OPOS consortium.

Release 1.1 is a superset of Release 1.01.

Section	Change
POS Keyboard	New device: Add information in several locations, plus POS Keyboard chapter and header file.
Second Page	Remove CompuServe reference.
Line Display CapCharacterSet property	Add values for Kana and Kanji.
Line Display CharacterSet property	Add Windows code page information.
POS Printer Data Characters and Escape Sequences	Add new sequences for: Feed and Paper cut Feed, Paper cut, and Stamp Feed lines Feed units Feed reverse Font typeface selection Reverse video Shading Scale horizontally Scale vertically Add width selection for underline sequence.
POS Printer:	Add the following properties and methods: CapCharacterSet property CapTransaction property ErrorLevel property ErrorString property FontTypefaceList property RecBarCodeRotationList property RotateSpecial property SlpBarCodeRotationList property TransactionPrint method ValidateData method
POS Printer CharacterSet property	Add Windows code page information.
POS Printer PrintBarCode method	Add information on effects of the RotateSpecial property.
POS Printer PrintImmediate and PrintNormal methods	Clarify the effects of Carriage Return and Line Feed.
Scanner ScanData property	Clarify the data that is present in this property.
OposDisp.h header file	Add CapCharacterSet values for Kana and Kanji.
OposPtr.h header file	Add CapCharacterSet values. Add ErrorLevel values. Add TransactionPrint <i>Control</i> values.

Release 1.2

Release 1.2 adds additional device classes, plus additional APIs based on requirements from various OPOS-US, OPOS-Japan, and OPOS-Europe members.

Release 1.2 is a superset of Release 1.1.

Section	Change
Cash Changer	New device: Add information in several locations, plus Cash Changer chapter and header file.
Tone Indicator	New device: Add information in several locations, plus Tone Indicator chapter and header file.
Several places	When a method has a <i>Timeout</i> parameter, added the constant OPOS_FOREVER as a value, and noted that OPOS_E_ILLEGAL can be returned.
First Two Pages	Update company names. Update copyright notices. Update web reference.
Introduction How an Application Uses an OPOS Control and Device Sharing Model	Explicitly state that a control may be simultaneously opened by many applications, but may be restricted in its functionality based on the Claim method.
Introduction Events	Add this section.
Introduction Input Model	Clarify the handling of error conditions. Add usage of AutoDisable and DataCount . Clarify the Error state exit conditions. Clarify when ClearInput is legal.
Introduction Output Model	Clarify the Error state conditions.
Introduction Result Code Model	Clarify the setting of ResultCodeExtended .
Common BinaryConversion , AutoDisable , and DataCount properties	Add these new properties. Throughout document, add to Summary sections for each device class. Throughout document, specify the BString properties and method parameters that are affected by BinaryConversion .
Common ControlObjectVersion and ServiceObjectVersion properties	Add compliance information when versions don't match.
Common FreezeEvents property	Clarify FreezeEvents role in delaying event firing.

- Common **ResultCodeExtended** property
Clarify the setting of **ResultCodeExtended**.
- Common **ClearInput** and **ClearOutput** methods
Correct return value information: May return one of three statuses.
- Common **Open** method
Correct return value information: **ResultCode** may not match method return value.
- Common **Release** method
Correct **DeviceEnabled** side effects: Only exclusive use devices are disabled during the **Release**.
- Common **StatusUpdateEvent** event
Clarify the initial firing of events at device enable.
- MICR **BankNumber**
Correct definition to digits 4-8 of the **TransitNumber**.
- MSR **ErrorReportingType**
Add this new property.
- MSR **ParseDecodeData**
Clarify inconsistency: Both **ParseDecodeData** and **ParseDecodedData** were used for this property.
- MSR **ErrorEvent**
Update for track level error notification.
- POS Keyboard General Information
Clarify the type of keyboards that may be a POS Keyboard.
- POS Keyboard **POSKeyData** property
Update definition of this property: A logical key value.
- POS Keyboard **CapKeyUp**, **EventTypes**, and **POSKeyEventType** properties
Add these new properties.
- POS Printer Escape Sequences
Clarify that escape sequences that are not OPOS sequences are passed through to the printer.
- POS Printer **CapConcurrentXxxYyy**
Clarify the interpretation of a FALSE value.
- POS Printer **XxxLineSpacing**
Clarify that line spacing includes the printed line height. Could have been interpreted as only the whitespace between each pair of lines.
- POS Printer **PrintBarcode**
Add list of symbologies.
- POS Printer **MapMode** and **XxxLetterQuality**
Clarified legal handling of **MapMode** when the printer supports half-dots.
Clarified potential impact on metrics when **XxxLetterQuality** is changed and **MapMode** is dots.
- POS Printer **SetBitmap**
Extend the bitmap number usage to allow the same bitmap to be used for both receipt and slip.

POS Printer TransactionPrint	Clarify when Busy and Extended statuses may be returned.
POS Printer ValidateData	Add “Underline” to the Illegal status section.
Scale Model	Correct to state the weight unit is defined by the device, and not settable by the application.
Scale CapDisplay	Add this new property.
Scale WeightUnit	Clarify inconsistency: Both WeightUnit and WeightUnits were used for this property.
Scanner ScanDataLabel and ScanDataType	Add these new properties.
Signature Capture “Real Time” feature	Add the new properties CapRealTimeData and RealTimeDataEnabled . Update various sections for real time operation.
Change History Release 1.1	Remove the compliance requirements for 1.1 Control Objects. This information was corrected and added to the common ControlObjectVersion and ServiceObjectVersion properties.
Opos.h header file	Add OPOS_FOREVER constant. Add BinaryConversion values.
OposMsr.h header file	Add ErrorReportingType values.
OposKbd.h header file	Add EventTypes values.
OposPtr.h header file	Remove PTR_RP_NORMAL_ASYNC. Add symbologies to match scanner.
OposScan.h header file	Add symbologies for ScanDataType .
Technical Details “Event Handlers”	Delete section. Much of the information was inaccurate, and the rest was merged into the new “Events” section in the first chapter.
Throughout	Correct various editing errors.

Release 1.3

Release 1.3 adds additional device classes, a few additional APIs, and some corrections.

Release 1.3 is a superset of Release 1.2.

Section	Change
First Two Pages	Update copyright notices. Update web reference.
General	Modify the use of the term event “firing.” Use “enqueue” and “deliver” appropriately to describe event firing.
Bump Bar	New device: Add information in several locations, plus Bump Bar chapter and header file.
Fiscal Printer	New device: Add information in several locations, plus Fiscal Printer chapter and header file.
PIN Pad	New device: Add information in several locations, plus PIN Pad chapter and header file.
Remote Order Display	New device: Add information in several locations, plus Remote Order Display chapter and header file.
Several places	Relax ErrorEvent “retry” response to allow its use with some input devices.
Introduction Events	Clarify effect of the top event being blocked.
Introduction Input Model	Add details concerning enqueueing and delivery of ErrorEvents . Add description of asynchronous input.
Introduction Device Power Reporting Model	Add this section.
Introduction OPOS Control Descriptions	Add CURRENCY data type.
Common CapPowerReporting, PowerNotify, PowerState properties	Add these properties here, plus... Add to the Summary section of each device.
Common ResultCode property	Generalize the meaning of OPOS_E_BUSY.
Common StatusUpdateEvent	Add power state reporting information. Change parameter name from <i>Data</i> to <i>Status</i> .
Every Device	Add power reporting properties to Summary section. Add StatusUpdateEvent support (if previously not reported). Add power reporting reference to existing StatusUpdateEvent descriptions.
MSR DecodeData	Add “raw format” description and column to track data table.
MSR ExpirationDate	Specify the format.

MSR TrackData	Specify that data excludes the sentinels and LRC. Add that decoding occurs when DecodeData is TRUE.
MSR ErrorEvent	Clarify that DataCount and AutoDisable are not relevant for MSR error events.
POSPrinter XxxLineChars	Add implementation recommendations.
POSPrinter PrintTwoNormal	Clarify the meaning of the <i>Stations</i> parameter, including the addition of new constants.
Scale	Add the following features: <ul style="list-style-type: none"> • Asynchronous input. Property AsyncMode. Method ClearInput, updates to ReadWeight. Events DataEvent and ErrorEvent. • Display of text. Properties CapDisplayText, MaxDisplayTextChars. Method DisplayText. • Price calculation. Properties CapPriceCalculating, SalesPrice, UnitPrice. • Tare weight. Properties CapTareWeight, TareWeight. • Scale zeroing. Property CapZeroScale. Method ZeroScale.
Tone Indicator Summary and General Information's Device Sharing	Consistently specify that Tone Indicator is a sharable device.
Opos.h header file	Add CapPowerReporting , PowerState , and PowerNotify properties. Add StatusUpdateEvent power reporting values.
OposPtr.h header file	Add new PrintTwoNormal station constants.
Throughout	Correct some editing errors.

Release 1.4

Release 1.4 adds one additional device class.

Release 1.4 is a superset of Release 1.3.

Section	Change
CAT	Added new device class, Credit Authorization Terminal which includes CAT chapter and header file. This device class was added at the request of OPOS-J and is used primarily in Japan. No other revisions were made to the version 1.3 of the OPOS specification.

Release 1.5

Release 1.5 is a superset of Release 1.4.

Release 1.5 adds 2 additional device classes.

Section	Change
First Two Pages	Update copyright notices. Update web references.
General	Replace Claim with ClaimDevice and Release with ReleaseDevice .
Introduction	Update references to OLE to ActiveX where appropriate.
Common OpenResult property	Add new property, plus add to the Summary section of each device.
Common ClaimDevice and ReleaseDevice	Name change plus update remarks.
Cash Changer	Added support for receipt of money functionality.
Cash Drawer	Added multi-drawer handling.
CAT	Added PaymentMedia property. The TransactionNumber property summary was changed to correctly show the type as String.
Fiscal Printer	Properties CountryCode , ErrorOutID , PrinterState , QuantityDecimalPlaces and QuantityLength have been updated to reflect the fact that they should be initialized after Open instead of Open , Claim and Enable . DuplicateReceipt : Corrected to show that is R/W. Added return values.
Line Display	Added DISP_CCS_UNICODE to CapCharacterSet and DISP_CS_UNICODE to CharacterSet to allow for devices that support the UNICODE character set.
MSR	Added Track4Data , CapTransmitSentinels and TransmitSentinels properties. Clarified support for JIS-II track data. DataEvent status: Added meaning for the high byte. ErrorEvent's ResultCodeExtended when ResultCode=OPOS_E_EXTENDED : Added meaning for the high byte.
PINPad	Added Track4Data property.
Point Card Reader Writer	New device: Add information in several locations, plus Point Card Reader Writer chapter and header file.

POS Keyboard	CapKeyUp : Corrected type from LONG to BOOL.
POS Power	New device: Add information in several locations, plus POS Power chapter and header file.
POS Printer	Added support for color printing (ink jet technology), printing both sides on the slip station and mark feed paper. Added PTR_CCS_UNICODE to CapCharacterSet and PTR_CS_UNICODE to CharacterSet to allow for devices that support the UNICODE character set.
Remote Order Display	Added ROD_CCS_UNICODE to CapCharacterSet and ROD_CS_UNICODE to CharacterSet to allow for devices that support the UNICODE character set.
Scale	Properties SalesPrice , TareWeight and UnitPrice have been updated to reflect the fact that they should be initialized after Open instead of Open , Claim and Enable . ErrorEvent : Added OPOS_ER_RETRY as a value response.
Signature Capture	Update Model to discuss AutoDisable implications. RealTimeDataEnabled : Clarify when this takes effect. DataEvent : Correct conditions when this event may be fired to include real-time data.
Tone Indicator	Properties AsyncMode , Tone1Pitch , Tone1Volume , Tone1Duration , Tone2Pitch , Tone2Volume , Tone2Duration and InterToneWait have been updated to reflect the fact that they should be initialized after Open instead of Open , Claim and Enable . Clarified handling of the Sound method when another application claims the device and calls the Sound method.
Opos.h header file	Add OpenResult property values.
Appendix C	Added header files for Point Card Reader Writer and POS Power. Updated other header files for devices modified in this specification.
Appendix D	Update System String information to include ATL usages.
Appendix E	Added this appendix for details on ClaimDevice and ReleaseDevice .

Release 1.6

Release 1.6 is a superset of Release 1.5.

Section	Change
Line Display	<p>Added the CapBlinkRate, CapCursorType, CapCustomGlyph, CapReadBack, CapReverse, BlinkRate, CursorType, CustomGlyphList, GlyphHeight and GlyphWidth properties.</p> <p>Added DefineGlyph and ReadCharacterAtCursor methods.</p> <p>Many updates in the General Information section.</p> <p>Updated the DisplayText and DisplayTextAt methods to include support for new attribute types for reverse video, DISP_DT_REVERSE and DISP_DT_BLINK_REVERSE.</p>
Fiscal Printer	<p>Added the CapAdditionalHeader, CapAdditionalTrailer, CapChangeDue, CapEmptyReceiptIsVoidable, CapFiscalReceiptStation, CapFiscalReceiptType, CapMultiContractor, CapOnlyVoidLastItem, CapPackageAdjustment, CapPostPreLine, CapSetCurrency, CapTotalizerType, ActualCurrency, AdditionHeader, AdditionalTrailer, ChangeDue, ContractorId, DateType, FiscalReceiptStation, FiscalReceiptType, MessageType, PostLine, PreLine and TotalizerType properties.</p> <p>Added the SetCurrency, PrintRecCash, PrintRecItemFuel, PrintRecItemFuelVoid, PrintRecPackageAdjustment, PrintRecPackageAdjustVoid, PrintRecRefundVoid, PrintRecSubtotalAdjustVoid and PrintRecTaxID methods.</p> <p>Added country support for Bulgaria and Romania.</p> <p>Many updates in the General Information section.</p> <p>Clarified the description of the CapPositiveAdjustment property.</p> <p>Updated the CountryCode, DayOpened and DescriptionLength properties to reflect additions to the specification.</p>

	Updated the EndFiscalReceipt , GetData , GetDate , PrintRecItem , PrintRecMessage , PrintRecNotPaid , PrintRecRefund , PrintRecSubtotal , PrintRecSubtotalAdjustment , PrintRecTotal , PrintRecVoid , PrintRecVoidItem , PrintZReport and SetHeaderLine methods to reflect additions to the specification.
	Updated the ErrorEvent event to reflect additions to the specification.
	Properties CountryCode , ErrorOutID , PrinterState , QuantityDecimalPlaces and QuantityLength have been updated to tone down strong language in the 1.5 revision that changes the “Initialized After” text.
Scale	Properties SalesPrice , TareWeight and UnitPrice have been updated to tone down strong language in the 1.5 revision that changes the “Initialized After” text
Tone Indicator	Properties AsyncMode , Tone1Pitch , Tone1Volume , Tone1Duration , Tone2Pitch , Tone2Volume , Tone2Duration and InterToneWait have been updated to tone down strong language in the 1.5 revision that changes the “Initialized After” text.
Appendix C	Added new constants for Fiscal Printer and Line Display updates.

Release 1.7

The change history above has been maintained to this point for historical reference.

No specific change history relative to the OPOS APG is maintained from this release forward. Refer to Appendix C for the change history details (if any) relative to this section.

Section 8: OPOS Control Programmer's Guide

Who Should Read This Section

This Section of the OPOS Appendix is targeted for the system developer who will write an OPOS Control.

This Section assumes that the reader understands the following:

- The POS peripheral device to be supported.
- ActiveX Control and Automation terminology and architecture.
- ActiveX Control Container development environments, such as Microsoft Visual Basic or Microsoft Visual C++. These will be required for testing the OPOS Control.
- A thorough knowledge of the OPOS models and APIs presented in the other sections of Appendix A, The OPOS Implementation Reference.

See the following Web sites for additional OPOS information:

Microsoft Retail Industry Page:

<http://www.microsoft.com/business/industry/ret/retoposoverview.asp>

Reference implementation – Common Control Objects:

<http://monroecs.com/opos.htm>

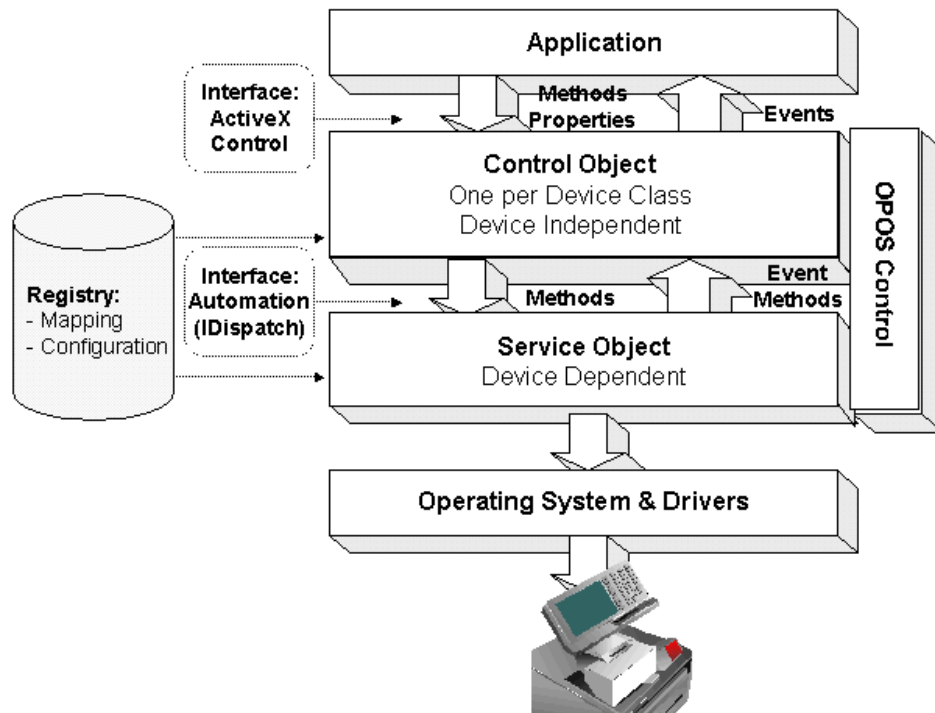
NRF-ARTS Standards Body

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

General OLE for Retail POS Control Model

OLE for Retail POS Controls adhere to the ActiveX Control specifications. They expose properties, events, and methods to a containing application. They specifically do not include a user interface, but rather rely exclusively upon the containing application for requests through methods and sometimes properties. Responses are given to the application through method return values and parameters, events, and properties.

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



OPOS Definitions

Device Class

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

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

Control Object or CO

A Control Object exposes a set of properties, methods, and events to an application for its device class. The OPOS Application Programmer's Guide describes these APIs.

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

Service Object or SO

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

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

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

A Service Object is typically implemented as a local in-proc server (in a DLL). In theory, it may also be implemented as a local out-proc server (in a separate executable process). However, we have found that, in practice, out-proc servers do not work well for OPOS Service Objects, and do not recommend their use.

OPOS Control or Control

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

Note - Service Object Implementation: Out-of-Process vs. In-Process Servers

In general, the primary difficulty in using out-proc automation servers arises when either of the two possible scenarios may occur:

- (A) The server is processing a COM function for the client application (such as when the client has called a Control's method) when another server calls a COM function in the client (such as when a Control's event is fired).
- (B) The server has called a COM function in a client application (such as when a Control's event is fired) when another client application calls a COM function in the server (such as when this client calls a Control's method).

The likelihood of these scenarios, especially (A), is greater for OPOS Service Objects since:

- Some OPOS methods require an indeterminately long time to be processed, such as the Cash Drawer **WaitForDrawerClose**.
- Some OPOS events may require an indeterminately long time to be processed, such as an **ErrorEvent** whose application handler waits for a user response to a dialog box.

The case where an OPOS event occurs from one service object while another service object is processing a method call or a property access then becomes probable.

These scenarios could be handled if both the client application and the out-proc server installed message filters (using the function **CoRegisterMessageFilter**), and the code for these filters dealt with these scenarios in an OPOS-prescribed manner. However, the default message filters for client environments such as Visual Basic and Visual C++ do not adequately handle the scenarios. Behavior varies from displaying a dialog and waiting for a user response (which is unacceptable for many POS operations) to generating an exception that terminates the client application (which is certainly unacceptable for POS applications). In addition, some environments do not provide a mechanism that easily allows an application to set up its custom message filter.

These issues simply do not exist when in-proc servers are used. Therefore, we recommend that they be used to implement service objects. This does, however, somewhat complicate sharing a Control between applications. Interprocess communication mechanisms, such as shared memory and named mutexes and events, may be used for this sharing.

If out-proc servers are used, then both the service object developer and the application developer will need to carefully implement message filters. The service object vendor should properly document this requirement to its application writers.

Interface Overview

A major OPOS objective is to provide general peripheral device APIs that can be applied to many vendors' peripherals. This leads to a requirement that any implementation of a Control Object be able to communicate with any vendor's Service Object. A straightforward example is with printers: Suppose a fast-food restaurant requires a local printer by one vendor and a remote kitchen printer by another vendor. Two instances of the printer CO will be required where each instance communicates with a different SO. The single CO must work with both vendors' SOs.

In order to define Control Objects that work across many vendors' Service Objects, the Control Object interfaces should be as generic and simple as possible. Therefore, the CO will maintain very little information and will, in general, perform the following three duties:

- Service Object coupling: Supervises a dispatch interface with a Service Object for the device.
- Methods and properties: Performs a pass-through of the application's method and property requests to the Service Object.
- Events: When a Service Object calls one of the special event request methods in the Control Object, the CO fires an appropriate event to the application.

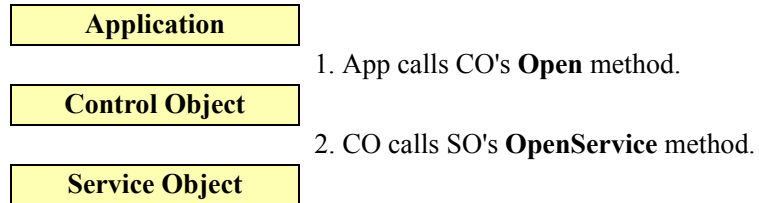
The various paths of communication between the application, Control Object, and Service Object are shown in the following sections.

Methods

An application initiates method calls to the OPOS Control.

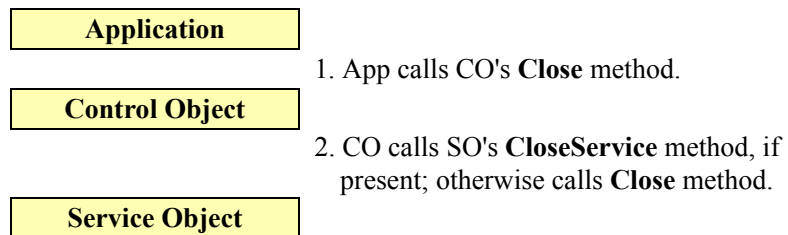
Open Method

The **Open** method is processed as follows:



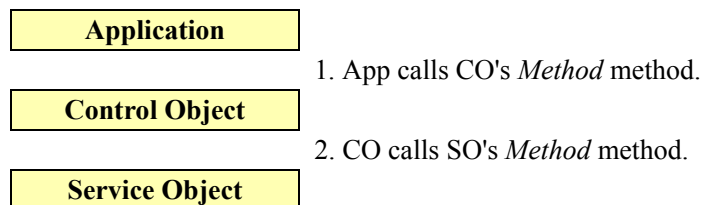
Close Method

The **Close** method is processed as follows:



Other Methods

All other methods are processed as follows, where *Method* represents the name of the method:



Properties

An application initiates property accesses to the OPOS Control.

String Properties

Gets and sets of string properties are processed as follows, where *StringProp* represents the name of the property:

Application

1. App accesses CO's *StringProp* property.

Control Object

2. If get, CO calls SO's **GetPropertyString** method, with an index that represents *StringProp*.

If set, CO calls SO's **SetPropertyString** method, with an index that represents *StringProp*.

The index values are specified in header files.

Service Object

LONG and BOOL Properties

Gets and sets of long and boolean properties are processed as follows, where *NumericProp* represents the name of the property:

Application

1. App accesses CO's *NumericProp* property.

Control Object

2. If get, CO calls SO's **GetPropertyNumber** method, with an index that represents *NumericProp*.

If set, CO calls SO's **SetPropertyNumber** method, with an index that represents *NumericProp*.

The index values are specified in header files.

Service Object

Other Property Types

Gets and sets of properties of any other type are processed as follows, where *Property* represents the name of the property:

Application

1. App accesses CO's *Property* property.

Control Object

2. If get, CO calls SO's **GetProperty** method.
If set, CO calls SO's **SetProperty** method.

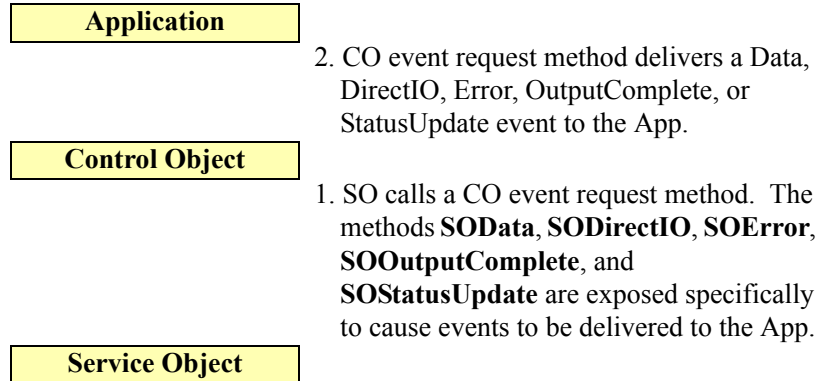
Service Object

Events

See “Events” on page A-11 in this Appendix for an overview of event handling.

The Service Object enqueues events, which are delivered to an application handler for the event.

The Service Object delivers events as follows:



Architecture: Firing an Event

A Service Object may need to fire an event for the following reasons:

- Method call or property set. A side effect of an application request to the control may cause an event to be fired.
Example: Assume that some data has been read and enqueued by the SO. When the application changes the **DataEventEnabled** property to TRUE, then the SO needs to deliver a **DataEvent**.
- Asynchronous activity. The Service Object will usually create one or more worker threads to monitor the device's input or output. The SO cannot rely upon the application to call OPOS methods or access OPOS properties on a regular basis to gain some processing time, so independently scheduled worker threads are a good solution. These threads may determine that an event needs to be fired.

Example: Assume that the **DataEventEnabled** property is TRUE, and that a worker thread is managing device input through a serial port. When the thread receives a data message, then the SO enqueues and needs to deliver a **DataEvent**.

When the SO needs to deliver an event, it calls a special event request method within the CO. The CO then delivers the event to the application.

Architectural Issue: Freezing Events by the Container

ActiveX control containers may freeze and unfreeze events by calling the **IOleControl::FreezeEvents** function. This is presented to a control written with MFC via the **COleControl::OnFreezeEvents** member function, or to an control

written with ATL via the **IOleControlImpl::FreezeEvents** member function. (One use of this feature is by the Visual Basic Common Dialog functions, which freeze events while the dialog is up.) When events have been frozen, a control should not deliver events. The Visual C++ documentation notes that the control may either discard events that occur during the freeze period, or it may buffer them for later delivery.

For OPOS Controls, enqueued events must be retained but not delivered while the container has frozen them. Then, when events are unfrozen by the container, the events may be delivered.

Each Service Object must support the method **COFreezeEvents**. The Control Object will call this method to freeze and unfreeze events.

Architectural Feature: Freezing Events by the Application

The application may wish to disable the arrival of events for a period of time. They may do this by setting the common boolean property **FreezeEvents** to TRUE.

The event freezing mechanism implemented for container-requested freezing is utilized to remember requests while the application has frozen them. Then, when the application sets the property to FALSE to unfreeze events, the events are delivered.

Summary of Event Firing

When a Service Object needs to deliver an event, it calls the appropriate event request method within the Control Object.

However, if events have been frozen due to a Control Object call to **COFreezeEvents** or due to the application setting the **FreezeEvents** property to TRUE, then the SO keep the event on its event queue. If the event is to be delivered from a worker thread, then this typically will be implemented by blocking the thread until events are unfrozen.

Control Object Responsibilities

The following sections describe the responsibilities of the Control Object. The Common Control Object is a reference implementation, whose source is available on the web.

Methods

The following sections describe the responsibilities of the Control Object methods.

If a device class does not support a common method (as specified by the device class Summary section in this document), then the Control Object should not define that method.

Since a Control Object must perform properly with any version of Service Object, as long as the major version numbers match, some bookkeeping must be performed in the Control Object. Specifically, the Control Object must not call methods that are not defined by a Service Object, or access properties that it does not define. In addition, it must perform additional management with the return values and **ResultCode**. (See the “OPOS Common Properties, Methods, and Events” on page A-24, “ControlObjectVersion” section for additional information.) The processing steps below assume that the Control Object defines a ResultCode flag to help manage version mismatch conditions.

Open Method

- If the Control Object is already open, then set **OpenResult** to OPOS_OR_ALREADYOPEN and return OPOS_E_ILLEGAL.
- If an empty device name has been passed, then set **OpenResult** to OPOS_OR_REGBADNAME and return OPOS_E_NOEXIST.
- Query the registry to find the Service Object that corresponds to this device class and device name. If the device class or device name is not found in the registry, then set **OpenResult** to OPOS_OR_REGBADNAME and return OPOS_E_NOEXIST.
- Load the Service Object for the device name. This requires (a) reading the device’s Programmatic ID from the registry, (b) converting it to a Class ID, (c) creating an instance of the Service Object, and (d) getting its IDispatch interface. If any of these are unsuccessful, then return OPOS_E_NOSERVICE. Set **OpenResult** to OPOS_OR_REGPROGID if (a) or (b) fails, or OPOS_OR_CREATE if (c) or (d) fails.

MFC (a) Use **RegQueryValueEx**. (b) Use **CLSIDFromProgID**. (c)-(d) Calling the **CreateDispatch** member function of an instance of the Service Object class, passing the Class ID from (b).

The Service Object class is generated by using the Visual C++ Class Wizard:

- Within the “OLE Automation” tab, push the “Add Class from an OLE TypeLib...” button. Then choose the .TLB file generated by a Service Object project.

- The Class Wizard will generate a **COleDispatchDriver** derivative, with member functions matching the OLE Automation methods exposed by the Service Object.

The Class Wizard will also generate an implementation of the member functions, which call **InvokeHelper** with fixed dispatch IDs. Since dispatch IDs depend upon the definition order of the automation methods, this implementation must be updated by the next step to allow for Service Objects that define the methods in a different order.

ATL (a) Use **RegQueryValueEx**. (b) Use **CLSIDFromProgID**.

(c) Use **CoCreateInstance**. (d) Use **QueryInterface** on the interface pointer returned by (c).

- Look up the dispatch IDs for all of the Service Object methods defined by the device class.

If any of the dispatch IDs defined in the initial version of the device class are not found in the Service Object, then close the dispatch interface, set **OpenResult** to **OPOS_OR_BADIF**, and return **OPOS_E_NOSERVICE**. (This ensures that the Service Object supports at least the minimum methods of a valid Service Object for the device class, before calling any of its methods.)

MFC Look up the dispatch IDs by calling the Service Object instance's **m_lpDispatch** → **GetIDsOfNames** function. Update the generated Service Object methods to pass these dispatch IDs to the **InvokeHelper** member function.

ATL Look up the dispatch IDs by calling the Service Object instance's **GetIDsOfNames** function. Save them for later use – they must be passed to the Service Object dispatch's **Invoke** function.

- Call the **OpenService** method of the Service Object, passing a device class string, a device name string, and the **IDispatch** pointer to the Control Object. If **OpenService** returns any result except **OPOS_SUCCESS**, then close the dispatch interface and return the **OpenService** result to the application. If the Service Object supports the method **GetOpenResult**, then call it and set **OpenResult** to its returned value; otherwise set **OpenResult** to **OPOS_OR_FAILEDOPEN**.

MFC The Control Object's dispatch pointer is accessed through its **GetIDispatch(FALSE)** member function.

ATL The Control Object's dispatch pointer is accessed by calling its **QueryInterface** function, requesting an **IDispatch** interface.

- Call the **GetPropertyNumber(PIDX_ServiceObjectVersion)** method of the Service Object to retrieve its version number. If the major version number is not one (1), then set **OpenResult** to **OPOS_OR_BADVERSION** and return **OPOS_E_NOSERVICE**.
- If any of the dispatch IDs for the methods that should be defined by the Service Object's version are not found, then:
 - call the Service Object's **CloseService** method if present, otherwise call its **Close** method,
 - close the dispatch interface,

- set **OpenResult** to OPOS_OR_BADIF,
- and return OPOS_E_NOSERVICE.

(This ensures that the Service Object supports all of the methods of a valid Service Object for the device class and version it claims to support. If the Service Object's version is newer than the Control Object, then the Control Object ensures that all of the methods for the Control Object's version are supported.)

- If all of the steps above are successful, then set an internal variable that shows that the Control Object is open, set **OpenResult** to OPOS_SUCCESS, and return OPOS_SUCCESS. Otherwise, the Control Object remains closed.

Close Method

- If the Control Object is closed, then return OPOS_E_CLOSED.
- If the Service Object supports the **CloseService** method, then call it. Otherwise, call its **Close** method.
- Set an internal variable that shows that the Control Object is closed.
- Release the Service Object.
 - **MFC** Call the **ReleaseDispatch** member function of the Service Object class.
 - **ATL** Call the Service Object dispatch pointer's **Release** member function.
- Return the result of the Service Object's **Close** method.

Other method calls

- If the Control Object is closed, then return OPOS_E_CLOSED.
- If the method was not defined in the Service Object's version of the device class, then:
 - Set the special ResultCode flag to show "version violation state".
 - Return OPOS_E_NOSERVICE.
- If the method is defined in the Service Object, then:
 - Pass the request down to the Service Object by calling the identically named Service Object method, using an identical list of parameters.
 - Set the special ResultCode flag to show "normal state".
 - Return the result of the Service Object method.

Properties

The Control Object processes property accesses as follows:

- The Control Object only maintains the properties **ControlObjectDescription**, **ControlObjectVersion**, and **OpenResult**. The Control Object will handle accesses to these properties directly, and return their value.
- If the Control Object is closed, then:
 - If setting a property, then return. (There is no means of informing the application that the set failed.)
 - If getting a property, then:
 - If the property is **State**, return OPOS_S_CLOSED.
 - If the property is **ResultCode**, return OPOS_E_CLOSED.
 - Otherwise, return a default property value:
FALSE for boolean.
Zero for numeric.
“[Error]” for string.
- If getting the property **ResultCode** and the special ResultCode flag is “version violation state”, then return OPOS_E_NOSERVICE.
- If the property is not supported by the version of the Service Object, then:
 - If setting a property, then set the special ResultCode flag to show “version violation state” and return.
 - If getting a property, then return the default property value.

If not one of the cases above...

- Set the internal ResultCode flag to show “normal state”.
- Pass down the request to the Service Object as follows.
 - If the property type is a 4-byte numeric value, including boolean and long, then call the Service Object's **GetPropertyNumber** or **SetPropertyNumber**. A parameter specifies the index of the property. These indices are established in the OPOS internal header files. In order to supply control objects for new devices, the writers of new device chapters may be requested to prepare the approximately 2-page data file used to define some of the key attributes of the device to the generator. In order to supply control objects for new devices, the writers of new device chapters may be requested to prepare the approximately 2-page data file used to define some of the key attributes of the device to the generator.
 - If the property type is string, then call the Service Object's **GetPropertyString** or **SetPropertyString**. A parameter specifies the index of the property. These indices are established in the OPOS internal header files.
 - If the property is any other type, then call the Service Object's get or set method for that property.

Events

The Service Object initiates events. The SO calls an event request method exposed by the Control Object.

The mapping of event request methods called by the Service Object into OPOS events is:

Event Request Methods	OPOS Event
SOData	DataEvent
SODirectIO	DirectIOEvent
SOError	ErrorEvent
SOOutputComplete	OutputCompleteEvent
SOStatusUpdate	StatusUpdateEvent

Upon receiving one of these event request methods, the Control Object delivers the appropriate event to the application. The Service Object thread will not regain control until the application event handler has completed.

Warning: These methods are only for use by the Service Object. Though accessible to the application, the application should not call them.

These five event request methods are defined on the following pages.

SOData

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

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

Remarks Requests the Control Object to deliver the event:

void DataEvent (LONG Status);

Called by the Service Object to deliver input data from the device to the application. The SO must not call **SOData** unless the **DataEventEnabled** property is TRUE. Just before calling **SOData**, the SO must change this property to FALSE, so that no further data events will be generated until the application sets this property back to TRUE. The actual input data is placed in one or more device class-specific properties.

SODirectIO

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

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

Remarks Requests the Control Object to deliver the event:

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

Called by the Service Object to communicate information directly to the application.

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

The Service Object must ensure that *pString* points to a valid system string, and then must free this string after **SODirectIO** returns.

SOError**Updated in Release 1.7**

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

Parameter	Description
<i>ResultCode</i>	Result code causing the error event. See “ResultCode Property” on page A-40 for values.
<i>ResultCodeExtended</i>	Extended result code causing the error event. See “ResultCodeExtended Property” on page A-42 for values.
<i>ErrorLocus</i>	Location of the error. See values below.
<i>pErrorResponse</i>	Pointer to the error event response. See values below.

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

Value	Meaning
OPOS_EL_OUTPUT	Error occurred while processing asynchronous output.
OPOS_EL_INPUT	Error occurred while gathering or processing event-driven input. No input data is available.

OPOS_EL_INPUT_DATA

Error occurred while gathering or processing event-driven input, and some previously buffered data is available.

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

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

Remarks Requests the Control Object to deliver the event:

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

Once **SOError** has been called, the Service Object must not request another error event until the error has been cleared. However, if an error with locus OPOS_EL_INPUT_DATA is fired and the event handler responds with OPOS_ER_CONTINUEINPUT, then the SO may fire another error event with OPOS_EL_INPUT after the enqueued input has been delivered.

SOOutputComplete

- Syntax** **void SOOutputComplete (LONG OutputID);**
- The *OutputID* parameter indicates the number of the asynchronous output request that has completed.
- Remarks** Requests the Control Object to deliver the event:
- void OutputCompleteEvent (LONG OutputID);**
- Called by the Service Object when a previously started asynchronous output request completes successfully.

SOStatusUpdate

- Syntax** **void SOStatusUpdate (LONG Data);**
- The *Data* parameter is for device class-specific data describing the type of status change.
- Remarks** Requests the Control Object to deliver the event:
- void StatusUpdateEvent (LONG Data);**
- Called by the Service Object when the SO needs to alert the application of a device status change.
- Examples include a change in the cash drawer position (open vs. closed) and a change in a POS printer sensor (form present vs. absent).

The following method is not related to event firing, but is a special purpose support method.

SOProcessID

- Syntax** **LONG SOProcessID();**
- Remarks** Return the process ID of the application in which the Control Object exists.
- The following method is provided to support local out-proc Service Objects. As noted in the introduction chapter, out-proc servers are not recommended for OPOS Service Objects. However, if a vendor successfully designs and implements such a Service Object, this method may be useful.
- For example, if a Service Object which supports Printer with MICR has allowed an application to **Claim** the printer, then it will want to restrict **Claim** of the MICR to the same application, since it is not reasonable for two applications to share such a device with such closely interacting classes.

Service Object Responsibilities and Implementation

Methods

The following common Service Object methods are defined for implementing corresponding Control Object methods. If a device class does not support a common method (as specified by the device class Summary section in the this document), then the Service Object should not define that method.

For each device class, additional methods are defined for each device specific method.

The general rules used to define the Service Object methods are:

- The Service Object method name is the same as the Control Object's method name.
- The parameters match those of the Control Object, both in order and type.

The only exceptions to these rules are the **OpenService**, **CloseService** (optional – may use **Close** instead), **GetOpenResult** (optional), and **COFreezeEvents** methods.

Note that these methods are always called through the Service Object's IDispatch interface.

For each of the methods below, syntax is shown for MFC as entered into the control's "Add Method" dialog, and for ATL as entered into the COM object's "Add Method to Interface" dialog.

CheckHealth

Syntax	MFC long CheckHealth (long <i>Level</i>); ATL HRESULT CheckHealth (long <i>Level</i> , [out, retval] long* <i>PRC</i>);
Remarks	Called to test the state of a device.

ClaimDevice / Claim

Syntax	MFC long ClaimDevice(long Timeout); long Claim(long Timeout); ATL HRESULT ClaimDevice(long Timeout, [out, retval] long* pRC); HRESULT Claim(long Timeout, [out, retval] long* pRC);
Remarks	Called to request exclusive access to the device. Release 1.0 – 1.4 Control Objects for these releases will only look for the Claim method. Release 1.5 and later A Control Object for this release will first look for the ClaimDevice method. If ClaimDevice is not present, then the Control Object looks for Claim .

ClearInput

Syntax	MFC long ClearInput(); ATL HRESULT ClearInput([out, retval] long* pRC);
Remarks	Called to clear all device input that has been enqueued.

ClearOutput

Updated in Release 1.7

Syntax	MFC long ClearOutput(); ATL HRESULT ClearOutput([out, retval] long* pRC);
Remarks	Called to clear all buffered output data, including all asynchronous output. Also, when possible, halts outputs that are in progress.

Close

Syntax	MFC long CloseService(); long Close(); ATL HRESULT CloseService([out, retval] long* pRC); HRESULT Close([out, retval] long* pRC);
Remarks	Called to release the device and its resources. Release 1.0 – 1.4 Control Objects for these releases will only look for the Close method. Release 1.5 and later A Control Object for this release will first look for the CloseService method. If CloseService is not present, then the Control Object looks for Close .

COFreezeEvents**Internal Control/Service Object Method**

Syntax **MFC** long COFreezeEvents(BOOL Freeze);
ATL HRESULT COFreezeEvents(VARIANT_BOOL Freeze,
 [out, retval] long* pRC);

The *Freeze* parameter is TRUE / VARIANT_TRUE when event firing must be frozen, and FALSE / VARIANT_FALSE when event firing is reenabled.

Remarks This method is for internal use by the Control Object.

The CO calls it in response to a container event freeze request to inform the SO of a change in the state of event firing. See “Architectural Issue: Freezing Events by the Container” on page A-96 for more information.

DirectIO

Syntax **MFC** long DirectIO(long *Command*, long* *pData*, BSTR* *pString*);
ATL HRESULT DirectIO(long *Command*, [in, out] long* *pData*,
 [in, out] BSTR* *pString*, [out, retval] long* *pRC*);

Remarks Call to communicate directly with the Service Object.

GetOpenResult**Internal Control/Service Object Method
Added in Release 1.5**

Syntax **MFC** long GetOpenResult();
ATL HRESULT GetOpenResult([out, retval] long* pRC);

Remarks This method is for internal use by the Control Object. It is optional.

If a Service Object’s **OpenService** method returns a status other than OPOS_SUCCESS, and it has implemented this method, then the Control Object calls this method to set its **OpenResult** property.

The Service Object may select one of the values given in the OPOS.H header file, or return a Service Object-specific value.

Return For MFC implementations, return one of the following values. For ATL implementations, store one of the following values at *pRC*, and return S_OK.

Value	Meaning
--------------	----------------

OPOS_OR_S_NOPORT	The Service Object tried to access an I/O port (for example, an RS232 port) during Open processing, but the port that is configured for the DeviceName is invalid or inaccessible.
------------------	---

As a general rule, an SO should refrain from accessing the physical device until the DeviceEnabled property is set to TRUE. But in some cases, it may require some access at **Open**; for instance, to dynamically determining the device type in order to set the DeviceName and DeviceDescription properties.

OPOS_ORN_NOTSUPPORTED

The Service Object does not support the specified device.

The SO has determined that it does not have the ability to control the device it is opening. This determination may be due to an inspection of the registry entries for the device, or dynamic querying of the device during **Open** processing.

OPOS_ORN_CONFIG Configuration information error.

Usually this is due to incomplete configuration of the registry, such that the SO does not have sufficient or valid data to open the device.

OPOS_ORN_SPECIFIC Errors greater than this value are service object-specific.

If the previous return values do not apply, then the SO may define additional **OpenResult** values. These values are Service Object-specific, but may be of value in these cases:

- 1) The Application logs or reports this error during debug and testing.
- 2) The Application adds SO-specific logic, to attempt to report more error conditions or to recover from them.

OpenService**Internal Control/Service Object Method**

Syntax **MFC** `long OpenService(LPCTSTR DeviceClass, LPCTSTR DeviceName, LPDISPATCH pDispatch);`
ATL `HRESULT OpenService(BSTR DeviceClass, BSTR DeviceName, IDispatch* pDispatch, [out, retval] long* pRC);`

Parameter	Description
<i>DeviceClass</i>	Contains the requested device class, which are given in the header file OPOS.HI. Examples are “CashDrawer” and “POSPrinter.”
<i>DeviceName</i>	Contains the Device Name to be managed by this Service Object. The relationship between the device name and physical devices is determined by entries within the operating system registry; a setup or configuration utility maintains these entries. (See the “Application Programmer’s Guide” appendix “OPOS Registry Usage.”)
<i>pDispatch</i>	Points to the Control Object’s dispatch interface, which contains the event request methods.

Remarks Call to open a device for subsequent I/O. The Control Object calls this method as part of its **Open** method processing.

The Service Object will use the *DeviceClass* and *DeviceName* parameters in inquiring the registry for additional device specific information.

The following steps assume that the Control Object is written using Visual C++ with MFC. Modify appropriately if another development environment is selected.

For MFC implementations, the *pDispatch* parameter may be used as follows:

- Attach to the Control Object’s **IDispatch** interface by passing the *pDispatch* **IDispatch** pointer to the **AttachDispatch** member function of an instance of a class that defines the Control Object’s event request methods.

This class is generated by using the Visual C++ Class Wizard:

- Within the “OLE Automation” tab, push the “Add Class from an OLE TypeLib...” button. Then choose the .TLB file generated by a Control Object project.
- The Class Wizard will generate a **COleDispatchDriver** derivative, with member functions matching the OLE Automation methods exposed by the Control Object.
- The Class Wizard will also generate an implementation of the member functions, which call **InvokeHelper** with fixed dispatch IDs. Since dispatch IDs depend upon the definition order of the automation methods, this implementation must be updated by the next step to allow for Control Objects that define the methods in a different order.
- The class definition and implementation should be updated to remove all of the non-event request methods.

- Look up the event request methods (such as **SOData**) by calling the Control Object instance's **m_lpDispatch** → **GetIDsOfNames** function. Update the generated Control Object methods to pass these dispatch IDs to the **InvokeHelper** member function.

For ATL implementations, the *pDispatch* parameter may be used directly to call IDispatch's **GetIDsOfNames** and **Invoke** functions. Alternatively, a **CComDispatchDriver** class instance may be created; its **Invoke1** and **InvokeN** functions may be used to call the event functions.

Note

The Service Object attaches back to the Control Object's dispatch pointer in order to access the event request methods within the CO. This implies the following two points:

- When the Control Object exposes the event request methods for access by the Service Object, these methods also become accessible by the application. The application, of course, should not call these methods.
 - The Service Object can access other methods and properties within the Control Object. This is not usually beneficial; however, the SO may wish to access the **ControlObjectDescription** or **ControlObjectVersion** to validate compatibility between itself and the CO.
-

Return

For MFC implementations, return one of the following values. For ATL implementations, store one of the following values at *pRC*, and return S_OK.

Value	Meaning
OPOS_SUCCESS	The Service Object is open. This does not tell the Control Object or Application that the device is online and functional. Rather, it states that the Service Object software is initialized, and ready to attempt device interaction when the DeviceEnabled property is set to TRUE.
<i>Other Values</i>	See "resultCode Property" on page A-40. Any return value except OPOS_SUCCESS is an Open failure, and will result in the Control Object shutting down the Service Object (by releasing its COM pointer) and passing this status to the Application. Since the APG defines meanings for OPOS_E_ILLEGAL and OPOS_E_NOEXIST, a Service Object should return one of these only if the failure is similar to one of these meanings. Otherwise, the Application may be misled.

Release 1.5 and later

On a failure, the Control Object will call the Service Object's **GetOpenResult** method, if present, to retrieve an additional status value.

ReleaseDevice / Release

Syntax **MFC** long ReleaseDevice();
 long Release();

 ATL HRESULT ReleaseDevice([out, retval] long* pRC);

Remarks Called to release exclusive access to the device.

Release 1.0 – 1.4

Control Objects for these releases will only look for the **Release** method.

Release 1.5 and later

A Control Object for this release will first look for the **ReleaseDevice** method. If **ReleaseDevice** is not present, then the Control Object looks for **Release**.

Note that ATL implementations cannot support the **Release** method (at least not without updating/overriding ATL classes).

Properties

The following methods are defined for getting and setting properties of the following types: 4-byte numeric and string.

For each method, the first parameter is:

LONG *PropIndex*

The values of *PropIndex* are specified in Opos.hi for the common properties. The values of class-specific properties are specified in the class-specific header files.

For robustness, the Service Object should validate the *PropIndex*. If an invalid value is found, then it could display a message box specifying the error, generate a debug exception, or produce another alert to the developer. This type of error should be found during development, testing, or staging prior to rollout to a customer, so the method of informing the user may be rather terse.

Note that these methods are always called through the Service Object's IDispatch interface.

GetPropertyNumber

Syntax **MFC** `long GetPropertyNumber(long PropIndex);`
ATL `HRESULT GetPropertyNumber(long PropIndex,`
 `[out, retval] long* pNumber);`

Return The current value of the LONG or BOOL / VARIANT_BOOL property.

For BOOL properties - VARIANT_BOOL COM IDL type - the Common Control Objects return a zero value as VARIANT_FALSE and a non-zero value as VARIANT_TRUE.

GetPropertyString

Syntax **MFC** `BSTR GetPropertyString(long PropIndex);`
ATL `HRESULT GetPropertyString(long PropIndex,`
 `[out, retval] BSTR* pString);`

Return The current value of the string property.

SetPropertyNumber

Syntax **MFC** `void SetPropertyNumber(long PropIndex, long Number);`
ATL `HRESULT SetPropertyNumber(long PropIndex, long Number);`

Remarks Sets the LONG or BOOL property to *Number*.

For BOOL properties - VARIANT_BOOL COM IDL type - the Common Control Objects pass a zero value as zero (0) and a non-zero value as one (1).

SetPropertyString

Syntax	MFC void SetPropertyString(long <i>PropIndex</i> , LPCTSTR <i>String</i>); ATL HRESULT SetPropertyString(long <i>PropIndex</i> , BSTR <i>String</i>);
Remarks	Sets the string property to <i>String</i> .

Note – Rationale for property get and set methods

Instead of using the four methods above, the Service Object interface could have defined distinct get methods for every property, plus set methods for writable properties.

Due to the large number of properties present in several Control Objects, however, the four methods above were chosen to reduce the amount of overhead and Service Object code.

Other Types: Not BSTR, LONG, or BOOL

If the Control defines properties of types that are not BStrings, LONGs, or BOOLEans, then the Service Object must define additional get and set methods for these properties.

If using Visual C++ with MFC, this is most easily accomplished through the Class Wizard by adding an Automation property.

Getting Other Property Types

Syntax	MFC <i>Type</i> GetPropertyName(); ATL HRESULT GetPropertyName([out, retval] <i>Type</i> * pProp);
---------------	---

Where *Type* is replaced by the property's type, and *PropertyName* is replaced by the property's name.

Return	The current value of the property.
---------------	------------------------------------

Example: If a property

CURRENCY SomeMoney;

is defined by the control, then the Service Object must define the method

MFC CURRENCY GetSomeMoney();

ATL HRESULT GetSomeMoney([out, retval] CURRENCY* pCY);

Setting Other Property Types

Syntax **MFC** void *SetPropertyName*(*Type value*);
 ATL HRESULT *SetPropertyName*(*Type value*);

Where *Type* is replaced by the property's type,
and *PropertyName* is replaced by the property's name.

Remarks Sets the property to *value*.

This method is only defined if the property *PropertyName* is a writable property.

Example: If a read/write property

CURRENCY SomeMoney;
is defined by the control, then the Service Object must define the method

MFC void *SetSomeMoney*(**CURRENCY** *NewMoneyValue*);
 ATL HRESULT *SetSomeMoney*(**CURRENCY** *NewMoneyValue*);

Events

A Service Object causes events to be fired by calling event methods in the Control Object. These methods are named:

SOData
 SODirectIO
 SOError
 SOOutputComplete
 SOStatusUpdate

They are described in "Control Object Responsibilities" on page A-98.

See the **OpenService** description on page A-110 for information about how to get the dispatch interface and dispatch IDs necessary for calling these functions.

OPOS CPG Change History

Release 1.01

Release 1.01 mostly adds clarifications and corrections, but the Line Display and Signature Capture chapters received substantive changes to correct deficiencies in their definition.

Section	Change
Second Page	Add name of Microsoft Web site for OPOS information.
Opos.hi header file	Remove HKEY_LOCAL_MACHINE from the root keys.
OposPtr.hi header file	Change ...Nearend to ...NearEnd . Change ...Barcode to ...BarCode .
OposScal.hi header file	Correct WeightUnits value from 1 to 2.
OposSig.hi header file	Change TotalVectors to TotalPoints . Change VectorArray to PointArray .

Release 1.1

Release 1.1 adds APIs based on requirements from OPOS-J, the Japanese OPOS consortium.

Section	Change
Second Page	Remove CompuServe reference.
Opos.hi header file	Add POS Keyboard values.
OposKbd.hi header file	New header file for POS Keyboard.
OposPtr.hi header file	Add the following properties: CapCharacterSet CapTransaction ErrorLevel ErrorString FontTypefaceList RecBarCodeRotationList RotateSpecial SlpBarCodeRotationList

Release 1.2

Release 1.2 adds additional device classes, plus additional APIs based on requirements from various OPOS-US, OPOS-Japan, and OPOS-Europe members.

Release 1.2 is a superset of Release 1.1.

Section	Change
First Two Pages	Update company names. Update copyright notices. Update web reference.
Introduction	Add discussion of out-proc and in-proc service objects.
Control Object Chapter	Update to include handling of version mismatch between the Control Object and Service Object. Add the method SOProcessID .
Opos.hi header file	Add Cash Changer and Tone Indicator. Add the following properties: AutoDisable BinaryConversion DataCount
OposChan.hi header file	New header file for Cash Changer.
OposMsr.hi header file	Add the property ErrorReportingType . Add the property ParseDecodedData , with value set the same as ParseDecodeData .
OposKbd.hi header file	Add the following properties: CapKeyUp EventTypes POSKeyEventProperties
OposScal.hi header file	Add the following properties: CapDisplay WeightUnit .
OposScan.hi header file	Add the following properties: ScanDataLabel ScanDataType
OposSig.hi header file	Add the following properties: CapRealTimeData RealTimeDataEnabled .
OposTone.hi header file	New header file for Tone Indicator.

Release 1.3

Release 1.3 adds additional device classes, a few additional APIs, and some corrections.

Release 1.3 is a superset of Release 1.2.

Section	Change
First Two Pages	Update copyright notices. Update web reference.
General	Modify the use of the term event “firing.” Use “enqueue” and “deliver” appropriately to describe event firing.
Control Object Chapter	SOError: Allow OPOS_ER_RETRY to be returned on input events if the Control supports it.
Service Object Chapter	Add descriptions of property methods that don’t fall into “4-byte number” or “string” types.
Opos.hi header file	Add Bump Bar, Fiscal Printer, PIN Pad, and Remote Order Display. Add the following properties: CapPowerReporting PowerNotify PowerState
OposBb.hi header file	New header file for Bump Bar
OposChan.hi header file	Correct the string indices to use PIDX_STRING instead of PIDX_NUMBER.
OposFptr.hi header file	New header file for Fiscal Printer
OposPPad.hi header file	New header file for PIN Pad
OposROD.hi header file	New header file for Remote Order Display
OposScal.hi header file	Add the following properties: CapDisplayText CapPriceCalculating CapTareWeight CapZeroScale AsyncMode MaxDisplayTextChars TareWeight
Several header files	Add validation functions for the first release containing the device.

Release 1.4

Release 1.4 adds 1 additional device class.

Release 1.4 is a superset of Release 1.3.

Section	Change
Opos.hi header file	Add CAT.
OposCat.hi header file	New header file for CAT.

Release 1.5

Release 1.5 adds 2 additional device classes.

Release 1.5 is a superset of Release 1.4.

Section	Change
First Two Pages	Update copyright notices. Update web references.
General	Update Claim and Release references to include ClaimDevice and ReleaseDevice information. Update references to OLE to ActiveX where appropriate. Generalize some references to MFC implementations, and add some ATL implementation information.
Control Object Responsibilities	Remove implementation details, and refer to the Common Control Objects.
Service Object GetOpenResult method	Add new method.
Opos.hi header file	Added Point Card Reader Writer and POS Power device categories.
OposCash.hi header file	Add CapMultiDrawerDetect property.
OposCat.hi header file	Add PaymentMedia property
OposCash.hi header file	Add DepositAmount , DepositStatus , DeviceStatus , CapDeposit , CapDepositDataEvent , CapPauseDeposit , CapRepayDeposit , DepositCashList , DepositCodeList and DepositCounts properties.
OposMSR.hi header file	Add CapTransmitSentinels , Track4Data and TransmitSentinels properties.

OposPcrw.hi header file	New header file for Point Card Reader Writer.
OposPpad.hi header file	Update to match the released 1.3 header file, then Remove the Amount property index – it isn't a string. Add Track4Data property.
OposPtr.hi header file	Add CapJrnCartridgeSensor , CapJrnColor , CapRecCartrdigeSensor , CapRecColor , CapRecMarkFeed , CapSlpBothSidesPrint , CapSlpCartridgeSensor , CapSlpColor , CartridgeNotify , JrnCartridgeState , JrnCurrentCartridge , RecCartridgeState , RecCurrentCartridge , SlpPrintSide , SlpCartridgeState , and SlpCurrentCartridge properties.
OposPwr.hi header file	New header file for POS Power.

Release 1.6

Release 1.6 is a superset of Release 1.5.

Section	Change
OposDisp.hi header file	Added CapBlinkRate , CapCursorType , CapCustomGlyph , CapReadBack , CapReverse , BlinkRate , CursorType , CustomGlyphList , GlyphHeight and GlyphWidth properties.
OposFptr.hi header file	Added CapAdditionalHeader , CapAdditionalTrailer , CapChangeDue , CapEmptyReceiptIsVoidable , CapFiscalReceiptStation , CapFiscalReceiptType , CapMultiContractor , CapOnlyVoidLastItem , CapPackageAdjustment , CapPostPreLine , CapSetCurrency , CapTotalizerType , ActualCurrency , AdditionHeader , AdditionalTrailer , ChangeDue , ContractorId , DateType , FiscalReceiptStation , FiscalReceiptType , MessageType , PostLine , PreLine and TotalizerType properties.

Release 1.7

The change history above has been maintained to this point for historical reference.

No specific change history relative to the OPOS CPG is maintained from this release forward. Refer to Appendix C for the change history details (if any) relative to this section.

Common Control Objects

As a combination of the personal effort of Curtiss Monroe plus as part of the commitment of his employer, Research Computer Services, Inc. (based in Dayton, Ohio) to the retail community, a complete set of OPOS control objects have been developed for public use. These have been dubbed the “Common Control Objects.”

These control objects are delivered as a reference implementation, believed to be correct and suitable for direct use by applications, but not warranted to be correct or to work with any vendor's Service Objects.

Features

- All OPOS controls are supported.
- ATL-based, using dual interfaces so that the app can access them via IDispatch or COM interfaces (of the form IOPOSCashDrawer, etc.).
- Built using Microsoft Visual C++. (Currently at Version 6.0, Service Pack 4.)
- Backward compatible with all releases of service objects. This means that they check for older SOs, and return the proper errors to the application if it accesses unsupported properties or methods.
- They have been tested with several major hardware vendors' Service Objects.
- Event firing logic supports well-behaved service objects that fire events from the thread that created the control, plus other service objects that fire them from other threads.
- Self-contained, requiring only standard OS DLLs. Specifically, they do not require MFC or ATL DLLs.
- Both MBCS and Unicode versions have been built and given limited testing. At this time, only the MBCS versions are being posted.
- Source code for all control objects is available.
- For future additions, it is easy to add new control objects or update old ones. A custom generator was developed that reads a data file for each control to be built. To add properties or methods, the procedure is (a) update the data files, (b) regenerate, and (c) build the resulting projects.

Availability and Future

Curtiss intends to maintain the control objects, and post corrections plus new releases at the site <http://www.monroeecs.com> as needed, for as long as he is affiliated with OPOS. Should he not be able to perform this function, then the OPOS Core Committee is authorized to do so.

In order to supply control objects for new devices, the writers of new device chapters may be requested to prepare the approximately 2-page data file used to define some of the key attributes of the device to the generator.

OPOS Internal Header Files

The header files are listed in alphabetical order. The mapping of device class name to header file name is as follows:

– General –	Opos.hi
Bump Bar	OposBb.hi
Cash Changer	OposChan.hi
Cash Drawer	OposCash.hi
CAT	OposCat.hi
Check Scanner	OposChk.hi
Coin Dispenser	OposCoin.hi
Fiscal Printer	OposFptr.hi
Hard Totals	OposTot.hi
Keylock	OposLock.hi
Line Display	OposDisp.hi
MICR	OposMicr.hi
Motion Sensor	OposMotion.hi
MSR	OposMsr.hi
PIN Pad	OposPpad.hi
Point Card Reader Writer	OposPerw.hi
POS Keyboard	OposKbd.hi
POS Power	OposPwr.hi
POS Printer	OposPtr.hi
Remote Order Display	OposRod.hi
Scale	OposScal.hi
Scanner	OposScan.hi
Signature Capture	OposSig.hi
Smart Card Reader Writer	OposScr.w.hi
Tone Indicator	OposTone.hi

Java for Retail POS — JavaPOS Implementation Reference

What Is Java for Retail POS?

Java for Retail POS (JavaPOS) provides for open POS device solutions for applications based on Java development technology. It is an implementation of the UnifiedPOS architecture that defines:

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

The Java for Retail POS standards committee was formed by a collection of retail vendors and end users, with a primary goal of providing device interfaces for the retail applications written in Java. Prior to version 1.7 of the UnifiedPOS and JavaPOS standards these documents were separate sets of documentation. This Appendix has been added to this UnifiedPOS Standard to provide guidance on how to implement services in a Java environment.

The JavaPOS committee will produce the following:

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

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

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

Benefits

The benefits of JavaPOS include:

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

Dependencies

Deployment of JavaPOS depends upon the following software components:

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

Relationship to OPOS

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

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

Therefore, most of the OPOS APIs were mapped into the Java language. The general translation rules are given in Section 3 of this Appendix, page B-88.

Who Should Read This Section

This section is targeted to both the application developer who will use JavaPOS Devices and the system developer who will write JavaPOS Devices.

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

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

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

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

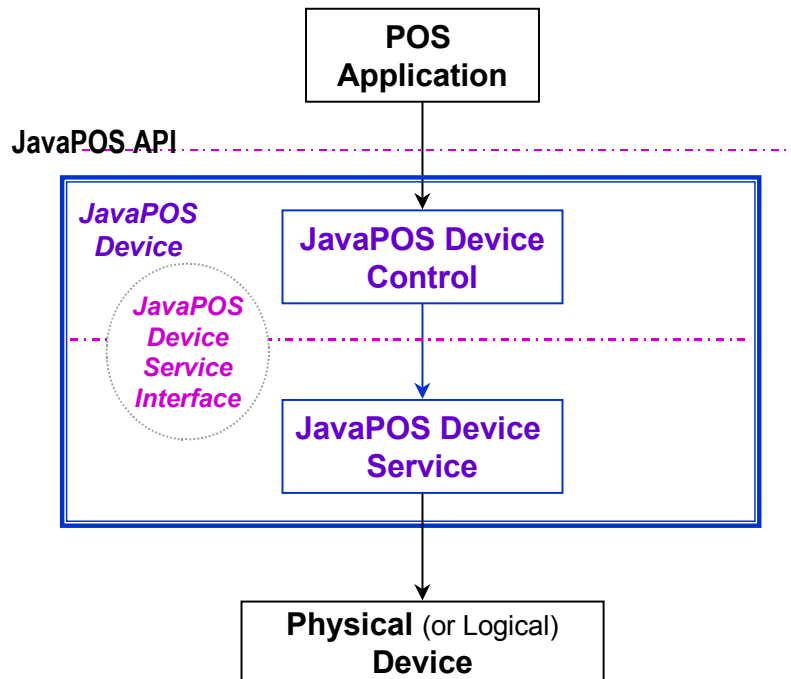
Appendix Overview

This appendix contains the following major sections:

Section Name	Developer Audience
What Is “Java for Retail POS?”	Application and System
Architectural Overview (page B-3)	Application and System
Device Behavior Models (page B-6)	Application and System
Classes and Interfaces (page B-30)	Application and System
Device Controls (page B-43)	System
Device Services (page B-52)	System

Architectural Overview

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



Architectural Components

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

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

Each JavaPOS Device is a combination of these components:

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

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

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

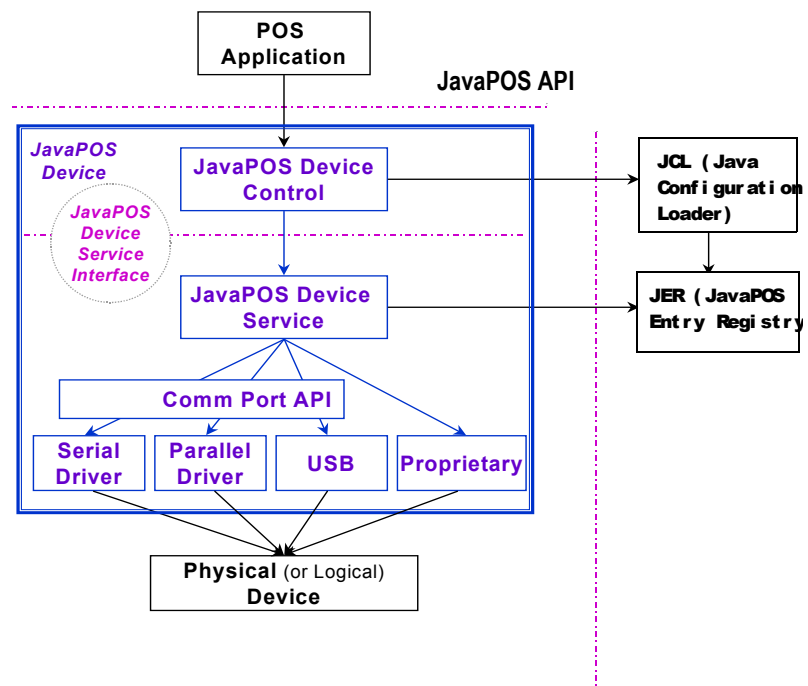
A set of Device Service classes can be implemented to support Physical Devices with multiple Device Categories.

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

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

Additional Layers and APIs

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



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

JavaPOS Development Environment

JavaPOS will use these packages:

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

Device Behavior Models

Introduction to Properties, Methods, and Events

An application accesses a JavaPOS Device via the JavaPOS APIs.

The three elements of JavaPOS APIs are:

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

To read a property value, use the method:

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

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

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

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

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

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

A JavaPOS method has the form:

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

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

Since JavaPOS uses Method names that are consistent with OPOS some Methods may appear to be Property getters/setters (for example, **setDate** page 355 in Fiscal Printer). BeanInfo classes are used to properly describe the Properties and Methods to provide clarification so that various vendors builder tools will properly function.

- **Events.** A JavaPOS Device may call back into the application via events. The application must specifically register for each event type that it needs to receive. JavaPOS events conform to the JavaBean event design pattern. See "Events" on page B-15 for further details.

Device Initialization and Finalization

Initialization

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

- Obtain a reference to a JavaPOS Device Control, either by creating a new instance or by accessing an existing one.
- Call Control methods to register for the events that the application needs to receive. (See “Events” on page B-15.)

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

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

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

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

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

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

Finalization

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

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

Summary

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

- Obtain a Device Control reference.
- Register for events (add listeners).
- Call the **open** method to instantiate a Device Service and link it to the Device Control.
- Call the **claim** method to gain exclusive access to the Physical Device. Required for exclusive-use Devices; optional for some sharable Devices. (See “Device Sharing Model” on page B-9 for more information).
- Set the **DeviceEnabled** property to true to make the Physical Device operational. (For sharable Devices, the Device may be enabled without first **claiming** it.)
- Use the device.
- Set the **DeviceEnabled** property to false to disable the Physical Device.
- Call the **release** method to release exclusive access to the Physical Device.
- Call the **close** method to unlink the Device Service from the Device Control.
- Unregister from events (remove listeners).

Device Sharing Model

JavaPOS Devices fall into two sharing categories:

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

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

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

Exclusive-Use Devices

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

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

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

Sharable Devices

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

Data Types

JavaPOS uses the following data types:

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

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

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

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

Exceptions

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

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

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

The parameters are:

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

ErrorCode

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

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

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

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

ErrorCodeExtended

The extended error code is set as follows:

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

Events

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

Event Class	Description	Supported When A Device Category Supports...
DataEvent	Input data has been placed into device class-category properties.	Event-driven input
ErrorEvent	An error has occurred during event-driven input or asynchronous output.	Event-driven input -or- Asynchronous output
OutputCompleteEvent	An asynchronous output has successfully completed.	Asynchronous output
StatusUpdateEvent	A change in the Physical Device's status has occurred. Release 1.3 and later: All devices may be able to report device power state. See "Device Power Reporting Model" on page B-24.	Status change notification
DirectIOEvent	This event may be defined by a Device Service provider for purposes not covered by the specification.	Always, for Service-specific use

Each of these events contains the following properties:

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

Chapter 1, "Events (UML interfaces)" on page 57, provides details about each of these events, including additional properties.

The Device Service must enqueue these events on an internally created and managed queue. All JavaPOS events are delivered in a first-in, first-out manner. (The only exception is that a special input error event is delivered early if some data events are also enqueued. See “Device Input Model” on page B-19.) Events are delivered by an internally created and managed Device Service thread. The Device Service causes event delivery by calling an event firing callback method in the Device Control, which then calls each registered listener's event method in the order in which they were added.

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

- The application has set the property **FreezeEvents** to true.
- The event type is a **DataEvent** or an input **ErrorEvent**, but the property **DataEventEnabled** is false. (See “Device Input Model” on page B-19.)

Rules for event queue management are:

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

Registering for Events

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

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

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

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

Event Delivery

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

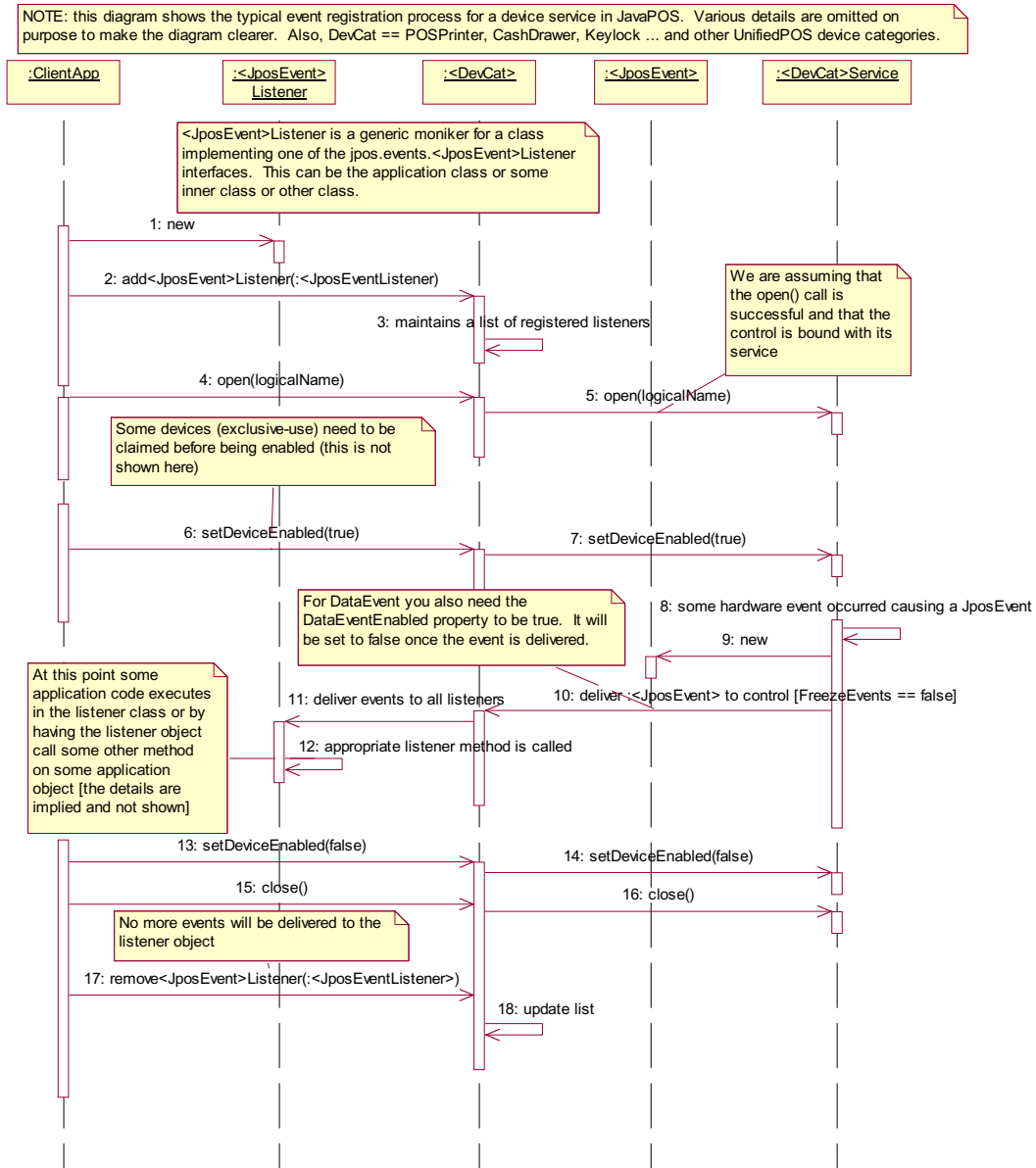
An application must not assume that events are delivered in the context of any particular thread. The JavaPOS Device delivers events on a privately created and managed thread. It is an application's responsibility to synchronize event processing with its threads as needed.

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

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

JavaPOS Event Registration Sequence Diagram *Added in Release 1.7*

The following sequence diagram shows how applications register for events with JavaPOS Controls, via classes implementing the JavaPOS event listener interface.



The delivery of events from a JavaPOS Service is almost always performed asynchronously to calls by clients; that is, once the clients have registered their <JposEvent>Listener objects with the Control, these listener objects will be called back – appropriate <jposEvent>Occurred() method – in a separate thread than the application thread. The event thread is usually a service thread that operates on an event queue, delivering all posted events from the queue to the Controls depending on whether the FreezeEvents property is true.

Device Input Model

The standard JavaPOS input model for exclusive-use devices is event-driven input. Event-driven input allows input data to be received after **DeviceEnabled** is set to true. Received data is queued as a **DataEvent**, which is delivered to an application as detailed in the “Events” (page B-15). If the **AutoDisable** property is true when data is received, then the JavaPOS Device will automatically disable itself, setting **DeviceEnabled** to false. This will inhibit the Device from enqueueing further input and, when possible, physically disable the device.

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

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

Error Handling

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

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

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

ErrorLocus	Description
JPOS_EL_INPUT_DATA	<p>Only delivered if the error occurred when one or more 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 JavaPOS Device, the operator can be immediately alerted to the error so that no further items are scanned until the error is resolved. Then, the application can process any backlog of previously scanned items before error recovery is performed.</p>
JPOS_EL_INPUT	<p>Delivered when an error has occurred and there is no data available.</p> <p>If some input data was buffered when the error occurred, then an ErrorEvent with the locus JPOS_EL_INPUT_DATA was delivered first, and then this error event is delivered after all DataEvents have been delivered.</p> <p>Note: This JPOS_EL_INPUT event is not delivered if: an JPOS_EL_INPUT_DATA event was delivered and the application event handler responded with a JPOS_ER_CLEAR.</p>

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

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

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

- The application returns from the JPOS_EL_INPUT **ErrorEvent**.
- The application returns from the JPOS_EL_INPUT_DATA **ErrorEvent**.
- The application calls the **clearInput** method.

Miscellaneous

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

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

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

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

Device Output Models

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

Synchronous Output

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

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

Asynchronous Output

Updated in Release 1.7

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

1. Buffers the request in program memory, for delivery to the Physical Device as soon as the Physical Device can receive and process it.
2. Sets the **OutputID** property to an identifier for this request.
3. Returns as soon as possible.

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

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

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

Error Handling

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

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

Miscellaneous

Updated in Release 1.7

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

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

Device Power Reporting Model

Added in JavaPOS Release 1.3, Updated in Release 1.8.

Applications frequently need to know the power state of the devices they use. Earlier Releases of JavaPOS had no consistent method for reporting this information. **Note:** This model is not intended to report Workstation or POS Terminal power conditions (such as “on battery” and “battery low”). Reporting of these conditions is now managed by the POSPower device category, see page 601.

Model

JavaPOS segments device power into three states:

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

In addition, one combination state is defined:

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

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

Note - Enabled/Disabled vs. Power States

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

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

Properties

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

- **CapPowerReporting** property. Identifies the reporting capabilities of the device. This property may be one of:
 - JPOS_PR_NONE. The Device Service cannot determine the state of the device. Therefore, no power reporting is possible.
 - JPOS_PR_STANDARD. The Device Service can determine and report two of the power states - OFF_OFFLINE (that is, off or offline) and ONLINE.
 - JPOS_PR_ADVANCED. The Device Service can determine and report all three power states - ONLINE, OFFLINE, and OFF.
- **PowerState** property. Maintained by the Device Service at the current power condition, if it can be determined. This property may be one of:
 - JPOS_PS_UNKNOWN
 - JPOS_PS_ONLINE
 - JPOS_PS_OFF
 - JPOS_PS_OFFLINE
 - JPOS_PS_OFF_OFFLINE
- **PowerNotify** property. The application may set this property to enable power reporting via **StatusUpdateEvents** and the **PowerState** property. This property may only be set before the device is enabled (that is, before **DeviceEnabled** is set to true). This restriction allows simpler implementation of power notification with no adverse effects on the application. The application is either prepared to receive notifications or doesn't want them, and has no need to switch between these cases. This property may be one of:
 - JPOS_PN_DISABLED
 - JPOS_PN_ENABLED

Power Reporting Requirements for DeviceEnabled

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

CapPowerReporting is not JPOS_PR_NONE, and
PowerNotify is JPOS_PN_ENABLED:

- When the Control changes from **DeviceEnabled** false to true, then begin monitoring the power state:
 - If the Physical Device is ONLINE, then:
 - PowerState** is set to JPOS_PS_ONLINE.
 - A **StatusUpdateEvent** is enqueued with its *Status* property set to JPOS_SUE_POWER_ONLINE.
 - If the Physical Device's power state is OFF, OFFLINE, or OFF_OFFLINE, then the Device Service may choose to fail the enable by throwing a **JposException** with error code JPOS_E_NOHARDWARE or JPOS_E_OFFLINE.

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

 - PowerState** is set to JPOS_PS_OFF, JPOS_PS_OFFLINE, or JPOS_PS_OFF_OFFLINE.
 - A **StatusUpdateEvent** is enqueued with its *Status* property set to JPOS_SUE_POWER_OFF, JPOS_SUE_POWER_OFFLINE, or JPOS_SUE_POWER_OFF_OFFLINE.
- When the Device changes from **DeviceEnabled** true to false, JavaPOS assumes that the Device is no longer monitoring the power state and sets the value of **PowerState** to JPOS_PS_UNKNOWN.

Device Information Reporting Model *Added in Release 1.8.*

POS Applications, as well as System Management agents, frequently need to monitor the current configuration and usage metrics of the various POS devices that are attached to the POS terminal.

Examples of configuration data are the device's *Serial Number*, *Firmware Version*, and *Connection Type*. Examples of usage data for the POSPrinter device are the *Number of Lines Printed*, *Number of Hours Running*, *Number of paper cuts*, etc. Examples of usage data for the Scanner device are the *Number of scans*, *Number of Hours Running*, etc. Examples of usage data for the MSR device are the *Number of successful swipes*, *Number of swipes resulting in errors*, *Number of Hours Running*, etc. See page 27 for examples of XML definitions of the device statistics accumulated per POS device category.

In some cases, the data may be accumulated and stored within the device itself. In other cases, the data may be accumulated by the Service and stored, possibly on the POS terminal or store controller.

In order for multiple applications (for example a POS application and a System Management application) to obtain statistics from the same device, proper care must be taken by both applications so that the device can be made accessible when required. This is done by using the **claim** and **setDeviceEnabled(true)** methods when access to a device is required and using the **setDeviceEnabled(false)** and **release** methods when access to the device is no longer needed. Coordination of device access via this mechanism is the responsibility of the applications themselves.

Statistics Reporting Properties and Methods

The UnifiedPOS device information reporting model adds the following common properties and methods across all device classes.

- **CapStatisticsReporting** property. Identifies the reporting capabilities of the device. When **CapStatisticsReporting** is false, then no statistical data regarding the device is available. This is equivalent to Services compatible with prior versions of the specification. When **CapStatisticsReporting** is true, then statistical data for the device is available.
- **CapUpdateStatistics** property. Defines whether gathered statistics (or some of them) can be reset/updated by the application. This property is only valid if **CapStatisticsReporting** is true. When **CapUpdateStatistics** is false, then none of the statistical data can be reset/updated by the application. Otherwise, when **CapUpdateStatistics** is true, then (some of) the statistical data can be reset/updated by the application.
- **resetStatistics** method. Can only be called if both **CapStatisticsReporting** and **CapUpdateStatistics** are true. This method resets one, some, or all of the resettable device statistics to zero.
- **retrieveStatistics** method. Can only be called if **CapStatisticsReporting** is true. This method retrieves one, some, or all of the accumulated statistics for the device.
- **updateStatistics** method. Can only be called if both **CapStatisticsReporting** and **CapUpdateStatistics** are true. This method updates one, some, or all of the resettable device statistics to the supplied values.

Device States

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

JPOS_S_CLOSED
JPOS_S_IDLE
JPOS_S_BUSY
JPOS_S_ERROR

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

- **State** is initially JPOS_S_CLOSED.
- **State** is changed to JPOS_S_IDLE when the **open** method is successfully called.
- **State** is set to JPOS_S_BUSY when the Device Service is processing output. The **State** is restored to JPOS_S_IDLE when the output has completed.
- The **State** is changed to JPOS_S_ERROR when an asynchronous output encounters an error condition, or when an error is encountered during the gathering or processing of event-driven input.

After the Device Service changes the **State** property to JPOS_S_ERROR, it enqueues an **ErrorEvent**. The properties of this event are the error code and extended error code, the locus of the error, and a modifiable response to the error. See Input Model, Error Handling on page B-20 and Output Model, Error Handling on page B-23 for further details.

Threads

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

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

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

Version Handling

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

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

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

Classes and Interfaces

Synopsis

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

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

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

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

Application

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

Device Control

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

Device Service

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

Helper Classes

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

Sample Class and Interface Hierarchies

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

Application Sample

“MyApplication” class hierarchy:

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

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

The “MyApplication” Application class also uses the following:

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

Device Control Sample

Scanner

Scanner class hierarchy:

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

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

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

POSPrinter

POSPrinter class hierarchy:

- **POSPrinterControl13**. Implement printer’s methods and properties and extends **POSPrinterControl12**.
- **EventCallbacks**. Derive an inner class to pass to Service so that it may generate events.

The **POSPrinter Control** class also uses the following:

- **JposConst** and **POSPrinterConst**. Use constants, either by fully qualified package names or by adding to the “implements” clause of the Device Control.
- **JposException**. Instance of this class is thrown when a method or property access fails.

Device Service Sample

“MyScannerService”

“MyScannerService” class hierarchy:

- **ScannerService12**. Implement scanner’s methods and properties.

The “MyScannerService” Service class also uses the following:

- **JposConst** and **ScannerConst**. Use constants, either by fully qualified package names or by adding to the “implements” clause of the Device Service.
- **DataEvent**. Instance of this class created as data is received. It is delivered to an application when the event delivery preconditions are met by calling the **fireDataEvent** method of the Control's derived **EventCallbacks** class.
- **ErrorEvent**. Instance of this class created when an error is detected while reading scanner data. It is delivered to an application when the event delivery preconditions are met by calling the **fireErrorEvent** method of the Control's derived **EventCallbacks** class.
- **JposException**. Instance of this class is thrown when a method or property access fails.

“MyPrinterService”

“MyPrinterService” class hierarchy:

- **POSPrinterService13.** Implement printer’s methods and properties and extends **POSPrinterService12.**

The “MyPrinterService” Service class also uses the following:

- **JposConst** and **POSPrinterConst.** Use constants, either by fully qualified package names or by adding to the “implements” clause of the Device Service.
- **ErrorEvent.** Instance of this class created when an error is detected while printing asynchronous data. It is delivered to an application when the event delivery preconditions are met by calling the **fireErrorEvent** method of the Control's derived **EventCallbacks** class.
- **OutputCompleteEvent.** Instance of this class created when an asynchronous output request completes. It is delivered to an application when the event delivery preconditions are met by calling the **fireOutputCompleteEvent** method of the Control's derived **EventCallbacks** class.
- **StatusUpdateEvent.** Instance of this class created when a printer status change is detected. It is delivered to an application when the event delivery preconditions are met by calling the **fireStatusUpdateEvent** method of the Control's derived **EventCallbacks** class.
- **JposException.** Instance of this class is thrown when a method or property access fails.

Sample Application Code

The following code snippet shows how to use a scanner.

```
//import ...;
import jpos.*;
import jpos.events.*;

public class MyApplication implements DataListener
{
    // Data listener's method to process incoming scanner data.
    public void dataOccurred(DataEvent e)
    {
        jpos.Scanner dc = (jpos.Scanner) e.getSource();
        String Msg = "Scanner DataEvent (Status=" + e.getStatus() +
            "\") received.";
        System.out.println (Msg);
        try {
            dc.setDataEventEnabled(true);
        } catch (JposException e){}
    }

    // Method to initialize the scanner.
    public void initScanner(String openName) throws jpos.JposException
    {
        // Create scanner instance and register for data events.
        jpos.Scanner myScanner1 = new jpos.Scanner();
        myScanner1.addDataListener(this);
        // Initialize the scanner. Exception thrown if a method fails.
        myScanner1.open(openName);
        myScanner1.claim(1000);
        myScanner1.setDeviceEnabled(true);
        myScanner1.setDataEventEnabled(true);
        //...Success! Continue doing work...
    }

    //...Other methods, including main...
}
```

Package Structure

The JavaPOS packages and files are as follows:

Note: The only difference between Release 1.3 and Release 1.4 of JavaPOS is the inclusion of the CAT device. No other technical changes were made. Therefore the JavaPOS packages and files for devices covered under Release 1.3 may be used for Release 1.4.

Additional device classifications of Point Card Reader Writer and POSPower were added in Release 1.5.

No new devices were added for Release 1.6, however additional functionality was added to some devices.

Additional device classifications of Check Scanner and Motion Sensor were added in Release 1.7.

Additional device classification of Smart Card Reader Writer was added in Release 1.8.

jpos

BaseControl.java
JposConst.java
JposException.java

CashChanger.java
CashChangerBeanInfo.java
CashChangerConst.java
CashChangerControl12.java

CashDrawer.java
CashDrawerBeanInfo.java
CashDrawerConst.java
CashDrawerControl12.java

CoinDispenser.java
CoinDispenserBeanInfo.java
CoinDispenserConst.java
CoinDispenserControl12.java

HardTotals.java
HardTotalsBeanInfo.java
HardTotalsConst.java
HardTotalsControl12.java

Keylock.java
KeylockBeanInfo.java
KeylockConst.java
KeylockControl12.java

LineDisplay.java
LineDisplayBeanInfo.java
LineDisplayConst.java
LineDisplayControl12.java

MICR.java
MICRBeanInfo.java
MICRConst.java
MICRControl12.java

MSR.java
MSRBeanInfo.java
MSRConst.java
MSRControl12.java

POSKeyboard.java
POSKeyboardBeanInfo.java
POSKeyboardConst.java
POSKeyboardControl12.java

POSPrinter.java
POSPrinterBeanInfo.java
POSPrinterConst.java
POSPrinterControl12.java

Scale.java
ScaleBeanInfo.java
ScaleConst.java
ScaleControl12.java

Scanner.java
ScannerBeanInfo.java
ScannerConst.java
ScannerControl12.java

SignatureCapture.java
SignatureCaptureBeanInfo.java
SignatureCaptureConst.java
SignatureCaptureControl12.java

ToneIndicator.java
ToneIndicatorBeanInfo.java
ToneIndicatorConst.java
ToneIndicatorControl12.java

New Peripheral Device Classes Added in Release 1.3

BumpBar.java	PINPad.java
BumpBarBeanInfo.java	PINPadBeanInfo.java
BumpBarConst.java	PINPadConst.java
BumpBarControl13.java	PINPadControl13.java
FiscalPrinter.java	RemoteOrderDisplay.java
FiscalPrinterBeanInfo.java	RemoteOrderDisplayBeanInfo.java
FiscalPrinterConst.java	RemoteOrderDisplayConst.java
FiscalPrinterControl13.java	RemoteOrderDisplayControl13.java

New Interfaces for existing Device Classes for Release 1.3

CashChangerControl13.java	MSRControl13.java
CashDrawerControl13.java	POSKeyboardControl13.java
CoinDispenserControl13.java	POSPrinterControl13.java
HardTotalsControl13.java	ScaleControl13.java
KeylockControl13.java	ScannerControl13.java
LineDisplayControl13.java	SignatureCaptureControl13.java
MICRControl13.java	ToneIndicatorControl13.java

New Peripheral Device Class Added in Release 1.4

CAT.java
 CATBeanInfo.java
 CATConst.java
 CATControl14.java

New Interfaces for existing Device Classes for Release 1.4

BumpBarControl14.java	MSRControl14.java
CashChangerControl14.java	PINPadControl14.java
CashDrawerControl14.java	POSKeyboardControl14.java
CoinDispenserControl14.java	POSPrinterControl14.java
FiscalPrinterControl14.java	RemoteOrderDisplayControl14.java
HardTotalsControl14.java	ScaleControl14.java
KeylockControl14.java	ScannerControl14.java
LineDisplayControl14.java	SignatureCaptureControl14.java
MICRControl14.java	ToneIndicatorControl14.java

New Peripheral Device Classes Added in Release 1.5

PointCardRW.java	POSPower.java
PointCardRWBeanInfo.java	POSPowerBeanInfo.java
PointCardRWConst.java	POSPowerConst.java
PointCardRWControl15.java	POSPowerControl15.java

New Interfaces for existing Device Classes for Release 1.5

BumpBarControl15.java	MSRControl15.java
CashChangerControl15.java	PINPadControl15.java
CashDrawerControl15.java	POSKeyboardControl15.java
CATControl15.java	POSPrinterControl15.java
CoinDispenserControl15.java	RemoteOrderDisplayControl15.java
FiscalPrinterControl15.java	ScaleControl15.java
HardTotalsControl15.java	ScannerControl15.java
KeylockControl15.java	SignatureCaptureControl15.java
LineDisplayControl15.java	ToneIndicatorControl15.java
MICRControl15.java	

New Interfaces for existing Device Classes for Release 1.6

BumpBarControl16.java	PINPadControl16.java
CashChangerControl16.java	PointCardRWControl16.java
CashDrawerControl16.java	POSKeyboardControl16.java
CATControl16.java	POSPowerControl16.java
CoinDispenserControl16.java	POSPrinterControl16.java
FiscalPrinterControl16.java	RemoteOrderDisplayControl16.java
HardTotalsControl16.java	ScaleControl16.java
KeylockControl16.java	ScannerControl16.java
LineDisplayControl16.java	SignatureCaptureControl16.java
MICRControl16.java	ToneIndicatorControl16.java
MSRControl16.java	

New Peripheral Device Classes Added in Release 1.7

CheckScanner.java	MotionSensor.java
CheckScannerBeanInfo.java	MotionSensorBeanInfo.java
CheckScannerConst.java	MotionSensorConst.java
CheckScannerControl17.java	MotionSensorControl17.java

New Interfaces for existing Device Classes for Release 1.7

BumpBarControl17.java	PINPadControl17.java
CashChangerControl17.java	PointCardRWControl17.java
CashDrawerControl17.java	POSKeyboardControl17.java
CATControl17.java	POSPowerControl17.java
CoinDispenserControl17.java	POSPrinterControl17.java
FiscalPrinterControl17.java	RemoteOrderDisplayControl17.java
HardTotalsControl17.java	ScaleControl17.java
KeylockControl17.java	ScannerControl17.java
LineDisplayControl17.java	SignatureCaptureControl17.java
MICRControl17.java	ToneIndicatorControl17.java
MSRControl17.java	

New Peripheral Device Class Added in Release 1.8

SmartCardRW.java
SmartCardRWBeanInfo.java
SmartCardRWConst.java
SmartCardRWControl18.java

New Interfaces for existing Device Classes for Release 1.8

BumpBarControl18.java	MSRControl18.java
CashChangerControl18.java	PINPadControl18.java
CashDrawerControl18.java	PointCardRWControl18.java
CATControl18.java	POSKeyboardControl18.java
CheckScannerControl18.java	POSPowerControl18.java
CoinDispenserControl18.java	POSPrinterControl18.java
FiscalPrinterControl18.java	RemoteOrderDisplayControl18.java
HardTotalsControl18.java	ScaleControl18.java
KeylockControl18.java	ScannerControl18.java
LineDisplayControl18.java	SignatureCaptureControl18.java
MICRControl18.java	ToneIndicatorControl18.java
MotionSensorControl18.java	

jpos.events

JposEvent.java

DataEvent.java

DataListener.java

DirectIOEvent.java

DirectIOListener.java

ErrorEvent.java

ErrorListener.java

OutputCompleteEvent.java

OutputCompleteListener.java

StatusUpdateEvent.java

StatusUpdateListener.java

jpos.services

BaseService.java	EventCallbacks.java
CashChangerService12.java	MSRService12.java
CashDrawerService12.java	POSKeyboardService12.java
CoinDispenserService12.java	POSPrinterService12.java
HardTotalsService12.java	ScaleService12.java
KeylockService12.java	ScannerService12.java
LineDisplayService12.java	SignatureCaptureService12.java
MICRService12.java	ToneIndicatorService12.java

New Peripheral Device Classes Added in Release 1.3

BumpBarService13.java	PINPadService13.java
FiscalPrinterService13.java	RemoteOrderDisplayService13.java

New Interfaces for Existing Device Classes for Release 1.3

CashChangerService13.java	MSRService13.java
CashDrawerService13.java	POSKeyboardService13.java
CoinDispenserService13.java	POSPrinterService13.java
HardTotalsService13.java	ScaleService13.java
KeylockService13.java	ScannerService13.java
LineDisplayService13.java	SignatureCaptureService13.java
MICRService13.java	ToneIndicatorService13.java

New Peripheral Device Classes Added in Release 1.4

CATService14.java

New Interfaces for Existing Device Classes for Release 1.4

BumpBarService14.java	MSRService14.java
CashChangerService14.java	PINPadService14.java
CashDrawerService14.java	POSKeyboardService14.java
CoinDispenserService14.java	POSPrinterService14.java
FiscalPrinterService14.java	RemoteOrderDisplayService14.java
HardTotalsService14.java	ScaleService14.java
KeylockService14.java	ScannerService14.java
LineDisplayService14.java	SignatureCaptureService14.java
MICRService14.java	ToneIndicatorService14.java

New Peripheral Device Classes Added in Release 1.5

PointCardRWService15.java	POSPowerService15.java
---------------------------	------------------------

New Interfaces for Existing Device Classes for Release 1.5

BumpBarService15.java	MSRService15.java
CashChangerService15.java	PINPadService15.java
CashDrawerService15.java	POSKeyboardService15.java
CATService15.java	POSPrinterService15.java
CoinDispenserService15.java	RemoteOrderDisplayService15.java
FiscalPrinterService15.java	ScaleService15.java
HardTotalsService15.java	ScannerService15.java
KeylockService15.java	SignatureCaptureService15.java
LineDisplayService15.java	ToneIndicatorService15.java
MICRService15.java	

New Interfaces for Existing Device Classes for Release 1.6

BumpBarService16.java	PINPadService16.java
CashChangerService16.java	PointCardRWService16.java
CashDrawerService16.java	POSKeyboardService16.java
CATService16.java	POSPowerService16.java
CoinDispenserService16.java	POSPrinterService16.java
FiscalPrinterService16.java	RemoteOrderDisplayService16.java
HardTotalsService16.java	ScaleService16.java
KeylockService16.java	ScannerService16.java
LineDisplayService16.java	SignatureCaptureService16.java
MICRService16.java	ToneIndicatorService16.java
MSRService16.java	

New Peripheral Device Classes Added in Release 1.7

CheckScannerService17.java MotionSensorService17.java

New Interfaces for Existing Device Classes for Release 1.7

BumpBarService17.java PINPadService17.java
CashChangerService17.java PointCardRWService17.java
CashDrawerService17.java POSKeyboardService17.java
CATService17.java POSPowerService17.java
CoinDispenserService17.java POSPrinterService17.java
FiscalPrinterService17.java RemoteOrderDisplayService17.java
HardTotalsService17.java ScaleService17.java
KeylockService17.java ScannerService17.java
LineDisplayService17.java SignatureCaptureService17.java
MICRService17.java ToneIndicatorService17.java
MSRService17.java

New Peripheral Device Classes Added in Release 1.8

SmartCardRWService18.java

New Interfaces for Existing Device Classes for Release 1.8

BumpBarService18.java MSRService18.java
CashChangerService18.java PINPadService18.java
CashDrawerService18.java PointCardRWService18.java
CATService18.java POSKeyboardService18.java
CheckScannerService18.java POSPowerService18.java
CoinDispenserService18.java POSPrinterService18.java
FiscalPrinterService18.java RemoteOrderDisplayService18.java
HardTotalsService18.java ScaleService18.java
KeylockService18.java ScannerService18.java
LineDisplayService18.java SignatureCaptureService18.java
MICRService18.java ToneIndicatorService18.java
MotionSensorService18.java

Device Controls

Note: This section is intended primarily for programmers who are creating JavaPOS Device Controls and Services.

Device Control Responsibilities

- Supporting the JavaPOS Device Interface for its category. This includes a set of properties, methods, and events.
- Managing the connection and interface to a Device Service.
- Forwarding most property accesses and method calls to the Device Service, and throwing exceptions when a property access or method call fails.
- Supporting add and remove event listener methods.
- Generating events to registered listeners upon command from the Device Service.
- Downgrading for older Device Service versions.

A Device Control is **not** responsible for:

- Managing multi-thread access to the Device Control and Service. An application must either access a Control from only one thread, or ensure that its threads synchronize sequences of requests as required to ensure that affected state and properties are maintained until the sequences have completed.
- Data buffering, including input and output data plus events. The Device Service manages all buffering and enqueueing.
- The device behavior/semantics and nuances that are specific to the functional control of the device.
- The loading functions that are to be contained in the `jpos.config/loader (JCL)`.

Device Service Management

The Device Control manages the connection to the Device Service. The Control calls upon the `jpos.config/loader` (JCL) to accomplish the connection and disconnection.

jpos.config/loader (JCL) and JavaPOS Entry Registry (JER)

The `jpos.config/loader` (JCL) along with the JavaPOS Entry Registry (JER) is used as the binding (configuration and loading) API that allows a JavaPOS control to bind to the correct JavaPOS service in a manner independent of the actual configuration mechanism. For POS applications, it represents a somewhat minimum (but extensible) functional equivalent of the “NT Registry” called the **JposEntryRegistry**.

All JavaPOS Device Controls that use this API and additional helpful reference material can be obtained on the JavaPOS website, <http://www.javapos.com>. In addition other standards information may be obtained from the <http://www.NRF-ARTS.org> website.

A reference open source implementation of the JCL is available on this website and maintained under the control of the JavaPOS technical committee. Included on the website is a functioning JCL with complete JavaDoc documentation, examples, sample code, a browser-based configuration editor and additional explanatory material.

A brief description of the JCL process is given below. However, for additional detailed information on the JCL one should consult the referenced web sites for the most up to date information.

jpos.config/loader (JCL) Characteristics

The `jpos.config/loader` is the name for the minimal set of classes (1) and interfaces (6) which are necessary to abstract into the JavaPOS specification. They provide for an independent way of configuring, loading and creating JavaPOS Device Services while maintaining the following important goals.

- Minimize the impact on existing controls
- Allow services to easily support multiple `jpos.config/loader` implementations
- Abstract as much as possible using Java interfaces to separate the JCL specification from its implementation
- Keep to a minimum the number of necessary classes and interfaces

The `jpos.config/loader` class/interfaces are added in two packages named `jpos.config` and `jpos.loader`. A `jpos` implementation is dependent upon the `jpos` and `jpos.loader` packages included in the `jpos.loader` class/interfaces, the `jpos.JposConst` interfaces and the `jpos.JposException` classes.

The `jpos.config/loader` specification contains 1 class and 6 interfaces. The single

class is the `jpos.loader.ServiceLoader` which bootstraps the implementation of the `jpos.config/loader` to be used in the JVM by creating the manager object (an instance of the `jpos.loader.JposServiceManager` interface). It also defaults to the simple `jpos.config/loader` implementation if no bootstrap is defined. The following table gives the name and a brief description of the class and interfaces that are involved.

Class or Interface	Name	Description
class	<code>jpos.loader.ServiceLoader</code>	This is the only class in the <code>jpos.config</code> and <code>jpos.loader</code> packages. It maintains a JposServiceManager instance (manager) which it uses to create a JposServiceConnection . The manager is created by looking for a Java property " <code>jpos.loader.serviceManagerClass</code> ". If this property is defined, then the class that it defines will be loaded and an instance of this class created as the manager (NOTE: this also assumes that the class implements <code>JposServiceManager</code> interface and has a 0-argument constructor). If the property is not defined then the "simple" JCL reference implementation manager is created (<code>jpos.loader.simple.SimpleServiceManager</code>).
interface	<code>jpos.loader.JposServiceManager</code>	This interface defines a manager used to create JposServiceConnection and allows access to the JposEntryRegistry .
interface	<code>jpos.loader.JposServiceConnection</code>	Defines a mediator between the service and the user of the service. The JavaPOS controls use this interface to connect to the service and then get the JposServiceInstance associated with the connection. Once disconnected the JposServiceInstance is no longer valid and a reconnect is necessary.
interface	<code>jpos.config.JposEntry</code>	Defines an interface for configuring a service. Properties can be added, queried, modified and removed. The JposServiceInstanceFactory uses the information in the object implementing this interface to create the current JposServiceInstance and configure it.
interface	<code>jpos.loader.JposEntryRegistry</code>	This interface defines a way to statistically and dynamically add known JposEntry objects to the system.
interface	<code>jpos.loader.JposServiceInstance</code>	Only interface required to be implemented by all JavaPOS services. It defines one method that is used to indicate to the service that the connection has been disconnected.
interface	<code>jpos.loader.JposServiceInstanceFactory</code>	Factory interface to create JposServiceInstance objects (i.e., the JavaPOS services). It is passed a JposEntry which it uses to create the correct service.

The configuration information is described as a set of properties in the **JposEntry**. These are entered as *<key, value>* pairs. The key is a String and the value is a Java Object of type: String, Integer, Long, Float, Boolean, Character or Byte (which are the String and primitive wrapper classes provided in the java.lang package). The following are two properties which must be defined by all the entries in the **JposEntry** in order for it to be considered valid.

Property Name	Property Type	Description
logicalName	String	This is the unique name that identifies this entry. The control uses this name to bind itself to the service.
serviceInstanceFactoryClass	String	Defines the factory class which should be used to create the service. This class must implement the jpos.loader.JposServiceInstanceFactory interface and it must have a default constructor.

All other properties are optionally provided or needed for the correct creation and initialization of the JavaPOS service. Note the service providers will most likely want to define their own set of properties and require them to be in the **JposEntry** in order to allow their **JposServiceFactory** to be used and their Device Service to be configured and loaded.

Future releases of the reference jpos.config/loader (JCL) might be modified to define a standard set of properties (in addition to the two mandated above) that all JavaPOS services would need to define.

Property and Method Forwarding

The Device Control must use the Device Service to implement all properties and methods defined by the JavaPOS Device Interface for a device category, with the following exceptions:

- **open** method.
- **close** method.
- **DeviceControlDescription** property. The Control returns its description.
- **DeviceControlVersion** property. The Control returns its version.
- **State** property. The Control forwards the request to the Service as shown in the following paragraphs. Any exception is changed to a return value of `JPOS_S_CLOSED`; an exception is never thrown to an application.

For all other properties and methods, the Device Control forwards the request to the identically named method or property of the Device Service. A template for set property and method request forwarding follows:

```
public void name(Parameters) throws JposException
{
    try
        service.name(Parameters);
    catch(JposException je)
        throw je;
    catch(Exception e)
        throw new JposException(JPOS_E_CLOSED,
            "Control not opened", e);
}
```

Similarly, a template for get property request forwarding is:

```
public Type name() throws JposException
{
    try
        return service.name();
    catch(JposException je)
        throw je;
    catch(Exception e)
        throw new JposException(JPOS_E_CLOSED,
            "Control not opened", e);
}
```

The general forwarding sequence is to call the Service to process the request, and return to the application if no exception occurs. If an exception occurs and the exception is **JposException**, rethrow it to the application.

Otherwise wrap the exception in a **JposException** and throw it. This should only occur if an **open** has not successfully linked the Service to the Control, that is, if the **service** field contains a null reference. (Any exceptions that occur while in the Service should be caught by it, and the Service should rethrow it as a **JposException**.) This allows the Control to set the message text to “Control not opened” with reasonable certainty.

Event Handling

Event Listeners and Event Delivery

An application must be able to register with the Device Control to receive events of each type supported by the Device, as well as unregister for these events. To conform to the JavaBean naming pattern for events, the registration methods have the form:

```
void addXxxListener(XxxListener l);  
void removeXxxListener(XxxListener l);
```

where *Xxx* is replaced by one of the event types: **Data**, **Error**, **OutputComplete**, **StatusUpdate**, or **DirectIO**.

An example add listener method is:

```
protected Vector dataListeners;  
public void addDataListener(DataListener l)  
{  
    synchronized(dataListeners)  
        dataListeners.addElement(l);  
}
```

When the Device Service requests that an event be delivered, the Control calls the event method of each listener that has registered for that event. (Typically, only one listener will register for each event type. However, diagnostic or other software may choose to listen, also.) The event methods have the form:

```
void xxxOccurred(XxxEvent e)
```

where *xxx* is replaced by: **data**, **error**, **outputComplete**, **statusUpdate**, or **directIO**.

Event Callbacks

The Device Service requests that an event be delivered by calling a method in a callback instance. This instance is created by the Control and passed to the Service in the **open** method.

The callback instance is typically created as an inner class of the Control. An example callback inner class is:

```
protected class ScannerCallbacks implements EventCallbacks
{
    public BaseControl getEventSource()
    {
        return (BaseControl)Scanner.this;
    }

    public void fireDataEvent(DataEvent e)
    {
        synchronized(Scanner.this.dataListeners)
            // deliver the event to all registered listeners
            for(int x = 0; x < dataListeners.size(); x++)
                ((DataListener)dataListeners.elementAt(x)).
                    dataOccurred(e);
    }

    public void fireDirectIOEvent(DirectIOEvent e)
    {
        //...Removed code similar to fireDataEvent...
    }

    public void fireErrorEvent(ErrorEvent e)
    {
        //...Removed code similar to fireDataEvent...
    }

    public void fireOutputCompleteEvent(OutputCompleteEvent e)
    {
    }

    public void fireStatusUpdateEvent(StatusUpdateEvent e)
    {
    }
}
```

Device Control Version Handling

The Device Control responsibilities given in the preceding sections “Device Service Management” and “Property and Method Forwarding” are somewhat simplified: They do not take into account version handling.

Both the Device Control and the Device Service have version numbers. Each version number is broken into three parts: Major, minor, and build. The major and minor portions indicate compliance with a release of the JavaPOS specifications. For example, release 1.4 compatibility is represented by a major version of one and a minor version of four. The build portion is set by the JavaPOS Device writer.

The JavaPOS version requirement is that a Device Control for a device category must operate and return reasonable results with any Device Service for that class, as long as its major version number matches the Service’s major version number.

In order to support this requirement, the following steps must be taken by the Control:

- **open** method. The Control must validate and determine the version of the Service, and save this version for later use (the “validated version”). The steps are as follows:

1. After connecting to the Device Service and obtaining its reference, determine the level of JavaPOS Service interface supported by the Service (the “interface version”). This test ensures that the Service complies with the property and method requirements of the interface.

For example, assume that the Scanner Control is at version 1.3. First attempt to cast the Service reference to the original release version, **ScannerService12**. If this succeeds, the “interface version” is at least 1.2; otherwise fail the **open**. Next, attempt to cast to **ScannerService13**. If this succeeds, the “interface version” is 1.3.

2. After calling the Service’s **open** method, get its **DeviceServiceVersion** property. If the major version does not match the Control’s major version, then fail the **open**.
3. At this point we know that some level of Service interface is supported, and that the major Control and Service versions match. Now determine the “validated version”:

```
if ( service_version <= interface_version )
{
    // The Service version may match the interface
    // version, or it may be less. The latter case may
    // be true for a Service that wraps or bridges to
    // OPOS software, because the Service may be able to
    // support a higher interface version, but
    // downgrades its reported Service version to that of
    // the OPOS software.
    // Remember the Services real version.
    validated_version = service_version;
}
else if ( service_version > interface_version )
```

```
{
    // The Service is newer than the Control.
    // Look at two subcases.
    if ( control_version == interface_version )
    {
        // The Service is newer than the Control, and it
        // supports all the Controls methods and
        // properties (and perhaps more that the Control
        // will not call).
        // Remember the maximum version that the Control
        // supports.
        validated_version = interface_version;
    }
    else if ( service_version > interface_version )
    {
        //... Fail the open!
        // The Service is reporting a version for which it
        // does not support all the required methods and
        // properties.
    }
}
```

- Properties and other methods. If an application accesses a property or calls a method supported by the Control's version but not by the "validated version" of the Service, the Control must throw a **JposException** with error code **JPOS_E_NOSERVICE**.

Device Services

Note: This section is intended primarily for programmers creating JavaPOS Device Controls and Services.

Device Service Responsibilities

A Device Service for a device category is responsible for:

- Supporting the JavaPOS Device Service Interface for its category. This includes a set of properties and methods, plus event generation and delivery.
- Implementing property accesses and method calls, and throwing exceptions when a property access or method call fails.
- Enqueuing events and delivering them (through calls to Device Control event callback methods) when the preconditions for delivering the event are satisfied.
- Managing access to the Physical Device.

The Device Service requires the `jpos.config/loader (JCL) JposEntry` object which contains all the configuration information.

Property and Method Processing

The Device Service performs the actual work for the property access and method processing. If the Service is successful in carrying out the request, it returns to the application. Otherwise, it must throw a **JposException**.

At the beginning of property and method processing, the Service will typically need to validate that an application has properly initialized the device before it is processed. If the device must first be claimed, the Service throws an exception with the error code `JPOS_E_CLAIMED` (if the device is already claimed by another JPOS Device) or `JPOS_E_NOTCLAIMED` (if the device is available to be claimed). If the device must first be enabled, then the Service throws an exception with the error code `JPOS_E_DISABLED`.

Some special cases are:

- **open** method. The Service must perform additional housekeeping and initialization during this method. Initialization will often include accessing the Java System Database (Release 1.4 and prior) or `JposEntryRegistry` (Release 1.5 and beyond) to obtain parameters specific to the Service and the Physical Device.
- **close** method. The Service releases all resources that were acquired during or after **open**.

Event Generation

The Device Service has the responsibility of enqueueing events and delivering them in the proper sequence. The Service must enqueue and deliver them one at a time, in a first-in, first-out manner. (The only exception is when a `JPOS_EL_INPUT_DATA` event must be delivered early on an input error because some data events are also enqueued.) Events are delivered by an internally created and managed Service thread. They are delivered by calling an event firing callback method in the Device Control, which then calls each registered listener's event method. (See “Event Handling” on page B-48.)

The following conditions cause event delivery to be delayed until the condition is corrected:

- The application has set the property **FreezeEvents** to true.
- The event type is a **DataEvent** or an input **ErrorEvent**, but the property **DataEventEnabled** is false. (See “Device Input Model” on page B-19.)

Rules on the management of the queue of events are:

- The JavaPOS Device may only enqueue new events while the Device is enabled.
- The Device may deliver enqueued events until the application calls the **release** method (for exclusive-use devices) or the **close** method (for any device), at which time any remaining events are deleted.
- For input devices, the **clearInput** method clears data and input error events.
- For output devices, the **clearOutput** method clears output error events.

Physical Device Access

The Device Service is responsible for managing the Physical Device. Often, this occurs by using a communications Port API (supplied or custom). At other times, the Service may need to use other device drivers or techniques to control the device.

The Java for Retail POS (JavaPOS) and OLE for Retail POS (OPOS) industry standard initiatives are intentionally similar in many respects.

Support for Java requires several differences from OPOS in architecture, but the JavaPOS committee agreed that the general model of OPOS device classes should be reused as much as possible.

In order to reuse as much of the OPOS device models as possible, the following sections detail the general mapping rules from OPOS to JavaPOS. A later section lists the deviations of JavaPOS APIs from OPOS.

API Mapping Rules

In most cases, OPOS APIs may be translated in a mechanical fashion to equivalent JavaPOS APIs. The exceptions to this mapping are largely due to differences in some string parameters.

Areas of data mapping include data types, methods and properties, and events.

JavaPOS Component Descriptions

The following sections are arranged as follows and provide detailed information on how an Application is expected to interface with a device covered under JavaPOS.

Section 1:

Describes the specific characteristics of the data types that JavaPOS uses as they relate to Java and a OS platform neutral implementation.

Section 2:

Provides interface descriptions for the properties, methods, and events specific to JavaPOS. For thorough description of these, one should consult the applicable chapters located in previous chapters in this document.

Section 3:

Compares the evolution of the JavaPOS from the OPOS standard and briefly describes some of the differences between the two implementations.

Section 4:

Provides the Change History previously contained in the JavaPOS Programmer's Guide.

Section 1: JavaPOS Data Types

Data Types

Data types are mapped from OPOS to JavaPOS as follows, with exceptions noted after the table:

Table 1:

OPOS Type	JavaPOS Type	Usage
BOOL	boolean	Boolean true or false.
BOOL *	boolean[1]	Modifiable boolean.
LONG	int	32-bit integer.
LONG *	int[1]	Modifiable 32-bit integer.
CURRENCY	long	64-bit integer. Used for currency values, with an assumed 4 decimal places.
CURRENCY *	long[1]	Modifiable 64-bit integer.
		<i>The string types are usually represented with the following mapping:</i>
BSTR	String	Text character string.
BSTR *	String[1]	Modifiable text character string.
		<i>For some APIs, the string types are represented in one of the following:</i>
	byte[]	Array of bytes. May be modified, but size of array cannot be changed. Often used when non-textual data is possible.
	Point[]	Array of points. Used by Signature Capture.
	Object	An object. This will usually be subclassed to provide a Device Service-specific parameter for directIO or DirectIOEvent .
<i>nls</i> (LONG)	<i>nls</i> (String)	Operating System National Language Data type.

Section 2: JavaPOS Interface Descriptions

Information in this section further defines the requirements of the UnifiedPOS for Java implementation. The common Properties, Methods, and Events are included to help transition from the UML given in Chapter 1 to the specifics for the Java Implementation on an Operating System that supports Java.

Next, tables are included that outline the specific programmatic examples for each of the device classifications and reference back to the UML for the respective devices.

The examples have been provided in Java and make no requirement of a specific OS in order to run.

JavaPOS Common Properties, Methods, and Events

Common Properties

Updated in Release 1.8

JavaPOS implementation specific definitions of the Common Properties.

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 }	1.2	1
CapPowerReporting	<i>int</i>	{ read-only }	1.3	
CapStatisticsReporting	<i>boolean</i>	{ read-only }	1.8	
CapUpdateStatistics	<i>boolean</i>	{ read-only }	1.8	
CheckHealthText	<i>String</i>	{ read-only }	1.0	
Claimed	<i>boolean</i>	{ read-only }	1.0	
DataCount	<i>int</i>	{ read-only }	1.2	1
DataEventEnabled	<i>boolean</i>	{ read-write }	1.0	1
DeviceEnabled	<i>boolean</i>	{ read-write }	1.0	
FreezeEvents	<i>boolean</i>	{ read-write }	1.0	
OutputID	<i>int</i>	{ read-only }	1.0	2
PowerNotify	<i>int</i>	{ read-write }	1.3	
PowerState	<i>int</i>	{ read-only }	1.3	
State	<i>int</i>	{ read-only }	1.0	
DeviceControlDescription	<i>String</i>	{ read-only }	1.0	
DeviceControlVersion	<i>int</i>	{ read-only }	1.0	
DeviceServiceDescription	<i>String</i>	{ read-only }	1.0	
DeviceServiceVersion	<i>int</i>	{ read-only }	1.0	
PhysicalDeviceDescription	<i>String</i>	{ read-only }	1.0	
PhysicalDeviceName	<i>String</i>	{ read-only }	1.0	

Usage Notes:

- 1.Used only with Devices that have Event Driven Input.
- 2.Used only with Asynchronous Output Devices.

Common Methods

Added in Release 1.8

JavaPOS implementation specific definitions of the Common Methods.

Methods (UML operations)

<i>Name</i>	<i>Version</i>
void open (String <i>logicalDeviceName</i>) throws JposException;	1.4
void close () throws JposException;	1.4
void claim (int <i>timeout</i>) throws JposException;	1.4
void release () throws JposException;	1.4
void checkHealth (int <i>level</i>) throws JposException;	1.4
void clearInput () throws JposException;	1.4
void clearOutput () throws JposException;	1.4
void directIO (int <i>command</i>, int[1] <i>data</i>, Object <i>object</i>) throws JposException;	1.4
void resetStatistics (String <i>statisticsBuffer</i>) throws JposException;	1.8
void retrieveStatistics (String[1] <i>statisticsBuffer</i>) throws JposException;	1.8
void updateStatistics (String <i>statisticsBuffer</i>) throws JposException;	1.8

JavaPOS Class Names

JavaPOS implementation specific definitions of the POS Device Categories' Class names.

UnifiedPOS Device Programmatic Names	JavaPOS Class Names
BumpBar	jpos.BumpBar
CashChanger	jpos.CashChanger
CashDrawer	jpos.CashDrawer
CAT	jpos.CAT
CheckScanner	jpos.CheckScanner
CoinDispenser	jpos.CoinDispenser
FiscalPrinter	jpos.FiscalPrinter
HardTotals	jpos.HardTotals
Keylock	jpos.Keylock
LineDisplay	jpos.LineDisplay
MICR	jpos.MICR
MotionSensor	jpos.MotionSensor
MSR	jpos.MSR
PINPad	jpos.PINPad
PointCardRW	jpos.PointCardRW
POSKeyboard	jpos.POSKeyboard
POSPower	jpos.POSPower
POSPrinter	jpos.POSPrinter
RemoteOrderDisplay	jpos.RemoteOrderDisplay
Scale	jpos.Scale
Scanner	jpos.Scanner
SignatureCapture	jpos.SignatureCapture
SmartCardRW	jpos.SmartCardRW
ToneIndicator	jpos.ToneIndicator

Properties

AutoDisable Property R/W

Type	boolean
Remarks	<p>If true, the Device 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 Device Service does not automatically disable the device when data is received.</p> <p>This property provides the application with an additional option for controlling the receipt of input data. If an application wants to receive and process only one input, or only one input at a time, then this property should be set to true. This property applies only to event-driven input devices.</p> <p>This property is initialized to false by the open method.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page B-12.

CapPowerReporting Property R

Added in Release 1.3

Type	int								
Remarks	<p>Identifies the reporting capabilities of the Device. It has one of the following values:</p> <table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>JPOS_PR_NONE</td> <td>The Device Service cannot determine the state of the device. Therefore, no power reporting is possible.</td> </tr> <tr> <td>JPOS_PR_STANDARD</td> <td>The Device Service can determine and report two of the power states - OFF_OFFLINE (that is, off or offline) and ONLINE.</td> </tr> <tr> <td>JPOS_PR_ADVANCED</td> <td>The Device Service can determine and report all three power states - OFF, OFFLINE, and ONLINE.</td> </tr> </tbody> </table> <p>This property is initialized by the open method.</p>	Value	Meaning	JPOS_PR_NONE	The Device Service cannot determine the state of the device. Therefore, no power reporting is possible.	JPOS_PR_STANDARD	The Device Service can determine and report two of the power states - OFF_OFFLINE (that is, off or offline) and ONLINE.	JPOS_PR_ADVANCED	The Device Service can determine and report all three power states - OFF, OFFLINE, and ONLINE.
Value	Meaning								
JPOS_PR_NONE	The Device Service cannot determine the state of the device. Therefore, no power reporting is possible.								
JPOS_PR_STANDARD	The Device Service can determine and report two of the power states - OFF_OFFLINE (that is, off or offline) and ONLINE.								
JPOS_PR_ADVANCED	The Device Service can determine and report all three power states - OFF, OFFLINE, and ONLINE.								
Errors	None.								

CapStatisticsReporting Property R

Added in Release 1.8

Type	boolean
Remarks	<p>If true, the device accumulates and can provide various statistics regarding usage; otherwise no usage statistics are accumulated. The information accumulated and reported is device specific, and is retrieved using the retrieveStatistics method.</p> <p>This property is initialized by the open method.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page B-12.
See Also	retrieveStatistics Method.

CapUpdateStatistics Property R*Added in Release 1.8*

Type	boolean
Remarks	<p>If true, the device statistics, or some of the statistics, can be reset to zero using the resetStatistics method, or updated using the updateStatistics method.</p> <p>If CapStatisticsReporting is false, then CapUpdateStatistics is also false.</p> <p>This property is initialized by the open method.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page B-12.
See Also	CapStatisticsReporting Property, resetStatistics Method, updateStatistics Method.

CheckHealthText Property R

Type	String
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 JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page B-12.

Claimed Property R

Type	boolean
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 JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page B-12.

DataCount Property R

Type	int
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	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page B-12.</p>

DataEventEnabled Property R/W

Type	boolean
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 Device Service until the condition is corrected.)</p> <p>If false, input data is enqueued for later delivery to the application. Also, if an input error occurs, the ErrorEvent is not delivered while this property is false.</p> <p>This property is initialized to false by the open method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page B-12.</p>

DeviceControlDescription Property R

Type	String
Remarks	<p>Holds an identifier for the Device Control and the company that produced it.</p> <p>A sample returned string is:</p> <pre>“POS Printer JavaPOS Control, (C) 1998 Epson”</pre> <p>This property is always readable.</p>
Errors	<p>None.</p>

DeviceControlVersion Property R**Type** **int****Remarks** Holds the Device Control version number.

Three version levels are specified, as follows:

Version Level	Description
Major	The “millions” place. A change to the JavaPOS major version level for a device class reflects significant interface enhancements, and may remove support for obsolete interfaces from previous major version levels.
Minor	The “thousands” place. A change to the JavaPOS minor version level for a device class reflects minor interface enhancements, and must provide a superset of previous interfaces at this major version level.
Build	The “units” place. Internal level provided by the Device Control developer. Updated when corrections are made to the Device Control implementation.

A sample version number is:

1002038

This value may be displayed as version “1.2.38”, and interpreted as major version 1, minor version 2, build 38 of the Device Control.

This property is always readable.

Errors None.**DeviceEnabled Property R/W****Type** **boolean****Remarks** If true, the device is in an operational state. If changed to true, then the device is brought to an operational state.

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.

Changing this property usually does not physically affect output devices. For consistency, however, the application must set this property to true before using output devices.

Release 1.3 and later: The Device’s power state may be reported while **DeviceEnabled** is true; See “Device Power Reporting Model” on page B-22 for details.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.

DeviceServiceDescription Property R

Type	String
Remarks	<p>Holds an identifier for the Device Service and the company that produced it.</p> <p>A sample returned string is:</p> <pre>TM-U950 Printer JPOS Service Driver, (C) 1998 Epson</pre> <p>This property is initialized by the open method.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page B-12.

DeviceServiceVersion Property R

Type	int								
Remarks	<p>Holds the Device Service version number.</p> <p>Three version levels are specified, as follows:</p> <table border="1"> <thead> <tr> <th>Version Level</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>Major</td> <td>The “millions” place. A change to the JavaPOS major version level for a device class reflects significant interface enhancements, and may remove support for obsolete interfaces from previous major version levels.</td> </tr> <tr> <td>Minor</td> <td>The “thousands” place. A change to the JavaPOS minor version level for a device class reflects minor interface enhancements, and must provide a superset of previous interfaces at this major version level.</td> </tr> <tr> <td>Build</td> <td>The “units” place. Internal level provided by the Device Service developer. Updated when corrections are made to the Device Service implementation.</td> </tr> </tbody> </table> <p>A sample version number is:</p> <pre>1002038</pre> <p>This value may be displayed as version “1.2.38”, and interpreted as major version 1, minor version 2, build 38 of the Device Service.</p> <p>This property is initialized by the open method.</p>	Version Level	Description	Major	The “millions” place. A change to the JavaPOS major version level for a device class reflects significant interface enhancements, and may remove support for obsolete interfaces from previous major version levels.	Minor	The “thousands” place. A change to the JavaPOS minor version level for a device class reflects minor interface enhancements, and must provide a superset of previous interfaces at this major version level.	Build	The “units” place. Internal level provided by the Device Service developer. Updated when corrections are made to the Device Service implementation.
Version Level	Description								
Major	The “millions” place. A change to the JavaPOS major version level for a device class reflects significant interface enhancements, and may remove support for obsolete interfaces from previous major version levels.								
Minor	The “thousands” place. A change to the JavaPOS minor version level for a device class reflects minor interface enhancements, and must provide a superset of previous interfaces at this major version level.								
Build	The “units” place. Internal level provided by the Device Service developer. Updated when corrections are made to the Device Service implementation.								
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page B-12.								

FreezeEvents Property R/W

Type	boolean
Remarks	<p>If true, events will not be delivered. Events will be enqueued until this property is set to false.</p> <p>If false, the application allows events to be delivered. If some events have been held while events were frozen and all other conditions are correct for delivering the events, then changing this property to false will allow these events to be delivered. An application may choose to freeze events for a specific sequence of code where interruption by an event is not desirable.</p> <p>This property is initialized to false by the open method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page B-12.</p>

OutputID Property R

Type	int
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 Device and are guaranteed to be unique among the set of outstanding asynchronous outputs. No other facts about the ID should be assumed.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page B-12.</p>

PowerNotify Property R/W**Added in Release 1.3****Type** **int****Remarks** Contains the type of power notification selection made by the Application. It has one of the following values:**Value****Meaning**

Value	Meaning
JPOS_PN_DISABLED	The Device Service will not provide any power notifications to the application. No power notification StatusUpdateEvents will be fired, and PowerState may not be set.
JPOS_PN_ENABLED	The Device 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 to JPOS_PN_DISABLED by the **open** method. This value provides compatibility with earlier releases.**Errors** A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page B-12.Some possible values of the exception's *ErrorCode* property are:**Value****Meaning**

Value	Meaning
JPOS_E_ILLEGAL	One of the following occurred:

The device is already enabled.

PowerNotify = JPOS_PN_ENABLED but
CapPowerReporting = JPOS_PR_NONE.

PowerState Property R***Added in Release 1.3***

Type	int												
Remarks	Identifies the current power condition of the device, if it can be determined. It has one of the following values:												
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>JPOS_PS_UNKNOWN</td> <td>Cannot determine the device's power state for one of the following reasons: CapPowerReporting = JPOS_PR_NONE; the device does not support power reporting. PowerNotify = JPOS_PN_DISABLED; power notifications are disabled. DeviceEnabled = false; Power state monitoring does not occur until the device is enabled.</td> </tr> <tr> <td>JPOS_PS_ONLINE</td> <td>The device is powered on and ready for use. Can be returned if CapPowerReporting = JPOS_PR_STANDARD or JPOS_PR_ADVANCED.</td> </tr> <tr> <td>JPOS_PS_OFF</td> <td>The device is powered off or detached from the POS terminal. Can only be returned if CapPowerReporting = JPOS_PR_ADVANCED.</td> </tr> <tr> <td>JPOS_PS_OFFLINE</td> <td>The device is powered on but is either not ready or not able to respond to requests. Can only be returned if CapPowerReporting = JPOS_PR_ADVANCED.</td> </tr> <tr> <td>JPOS_PS_OFF_OFFLINE</td> <td>The device is either off or offline. Can only be returned if CapPowerReporting = JPOS_PR_STANDARD.</td> </tr> </tbody> </table> <p>This property is initialized to JPOS_PS_UNKNOWN by the open method. When PowerNotify is set to enabled and DeviceEnabled is true, then this property is updated as the Device Service detects power condition changes.</p>	Value	Meaning	JPOS_PS_UNKNOWN	Cannot determine the device's power state for one of the following reasons: CapPowerReporting = JPOS_PR_NONE; the device does not support power reporting. PowerNotify = JPOS_PN_DISABLED; power notifications are disabled. DeviceEnabled = false; Power state monitoring does not occur until the device is enabled.	JPOS_PS_ONLINE	The device is powered on and ready for use. Can be returned if CapPowerReporting = JPOS_PR_STANDARD or JPOS_PR_ADVANCED.	JPOS_PS_OFF	The device is powered off or detached from the POS terminal. Can only be returned if CapPowerReporting = JPOS_PR_ADVANCED.	JPOS_PS_OFFLINE	The device is powered on but is either not ready or not able to respond to requests. Can only be returned if CapPowerReporting = JPOS_PR_ADVANCED.	JPOS_PS_OFF_OFFLINE	The device is either off or offline. Can only be returned if CapPowerReporting = JPOS_PR_STANDARD.
Value	Meaning												
JPOS_PS_UNKNOWN	Cannot determine the device's power state for one of the following reasons: CapPowerReporting = JPOS_PR_NONE; the device does not support power reporting. PowerNotify = JPOS_PN_DISABLED; power notifications are disabled. DeviceEnabled = false; Power state monitoring does not occur until the device is enabled.												
JPOS_PS_ONLINE	The device is powered on and ready for use. Can be returned if CapPowerReporting = JPOS_PR_STANDARD or JPOS_PR_ADVANCED.												
JPOS_PS_OFF	The device is powered off or detached from the POS terminal. Can only be returned if CapPowerReporting = JPOS_PR_ADVANCED.												
JPOS_PS_OFFLINE	The device is powered on but is either not ready or not able to respond to requests. Can only be returned if CapPowerReporting = JPOS_PR_ADVANCED.												
JPOS_PS_OFF_OFFLINE	The device is either off or offline. Can only be returned if CapPowerReporting = JPOS_PR_STANDARD.												
Errors	None.												

PhysicalDeviceDescription Property R

Type	String
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 JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page B-12.

PhysicalDeviceName Property R

Type	String
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 JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page B-12.

State Property R

Type	int										
Remarks	<p>Holds the current state of the Device. It has one of the following values:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>JPOS_S_CLOSED</td> <td>The Device is closed.</td> </tr> <tr> <td>JPOS_S_IDLE</td> <td>The Device is in a good state and is not busy.</td> </tr> <tr> <td>JPOS_S_BUSY</td> <td>The Device is in a good state and is busy performing output.</td> </tr> <tr> <td>JPOS_S_ERROR</td> <td>An error has been reported, and the application must recover the Device to a good state before normal I/O can resume.</td> </tr> </tbody> </table> <p>This property is always readable.</p>	Value	Meaning	JPOS_S_CLOSED	The Device is closed.	JPOS_S_IDLE	The Device is in a good state and is not busy.	JPOS_S_BUSY	The Device is in a good state and is busy performing output.	JPOS_S_ERROR	An error has been reported, and the application must recover the Device to a good state before normal I/O can resume.
Value	Meaning										
JPOS_S_CLOSED	The Device is closed.										
JPOS_S_IDLE	The Device is in a good state and is not busy.										
JPOS_S_BUSY	The Device is in a good state and is busy performing output.										
JPOS_S_ERROR	An error has been reported, and the application must recover the Device to a good state before normal I/O can resume.										
Errors	None.										

Methods

checkHealth Method

Syntax **void checkHealth (int *level*) throws JposException;**

The *level* parameter indicates the type of health check to be performed on the device. The following values may be specified:

Value	Meaning
JPOS_CH_INTERNAL	Perform a health check that does not physically change the device. The device is tested by internal tests to the extent possible.
JPOS_CH_EXTERNAL	Perform a more thorough test that may change the device. For example, a pattern may be printed on the printer.
JPOS_CH_INTERACTIVE	Perform an interactive test of the device. The supporting Device Service will typically display a modal dialog box to present test options and results.

Remarks Tests the state of a device.

A text description of the results of this method is placed in the **CheckHealthText** property. The health of many devices can only be determined by a visual inspection of these test results.

This method is always synchronous.

Errors A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page B-12.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	The specified health check level is not supported by the Device Service.

claim Method

Syntax **void claim (int *timeout*) throws JposException;**

The *timeout* parameter gives the maximum number of milliseconds to wait for exclusive access to be satisfied. If zero, then immediately either returns (if successful) or throws an appropriate exception. If JPOS_FOREVER (-1), the method waits as long as needed until exclusive access is satisfied.

Remarks Requests exclusive access to the device. Many devices require an application to claim them before they can be used.

When successful, the **Claimed** property is changed to true.

Errors A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page B-12.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	This device cannot be claimed for exclusive access, or an invalid <i>timeout</i> parameter was specified.
JPOS_E_TIMEOUT	Another application has exclusive access to the device, and did not relinquish control before <i>timeout</i> milliseconds expired.

clearInput Method

Syntax **void clearInput () throws JposException;**

Remarks Clears all device input that has been buffered.

Any data events or input error events that are enqueued – usually waiting for **DataEventEnabled** to be set to true and **FreezeEvents** to be set to false – are also cleared.

Errors A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page B-12.

clearOutput Method

Updated in Release 1.7

Syntax **void clearOutput () throws JposException;**

Remarks Clears all buffered output data, including all asynchronous output. Also, when possible, halts outputs that are in progress.

Any output error events that are enqueued – usually waiting for **FreezeEvents** to be set to false – are also cleared.

Errors A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page B-12.

close Method**Syntax** **void close () throws JposException;****Remarks** Releases the device and its resources.

If the **DeviceEnabled** property is true, then the device is disabled.

If the **Claimed** property is true, then exclusive access to the device is released.

Errors A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page B-12.**directIO Method****Syntax** **void directIO (int *command*, int[] *data*, Object *object*) throws JposException;**

Parameter	Description
<i>command</i>	Command number whose specific values are assigned by the Device Service.
<i>data</i>	An array of one modifiable integer whose specific values or usage vary by <i>command</i> and Device Service.
<i>object</i>	Additional data whose usage varies by <i>command</i> and Device Service.

Remarks Communicates directly with the Device Service.

This method provides a means for a Device Service to provide functionality to the application that is not otherwise supported by the standard Device Control for its device category. Depending upon the Device Service’s definition of the command, this method may be asynchronous or synchronous.

Use of this method will make an application non-portable. The application may, however, maintain portability by performing **directIO** calls within conditional code. This code may be based upon the value of the **DeviceServiceDescription**, **PhysicalDeviceDescription**, or **PhysicalDeviceName** property.

Errors A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page B-12.

open Method

Syntax **void open(String *logicalDeviceName*) throws JposException;**

The *logicalDeviceName* parameter specifies the device name to open.

Remarks Opens a device for subsequent I/O.

The device name specifies which of one or more devices supported by this Device Control should be used.

In Controls from version 1.4 and prior, The *logicalDeviceName* must exist in the Java System Database (JSD) for this device category so that its relationship to the physical device can be determined. Entries in the JSD are created by a setup or configuration utility.

In Controls from version 1.5 and beyond, The *logicalDeviceName* must exist in the **JposEntryRegistry** for this device category so that its relationship to the physical device can be determined. JposEntry objects in the registry are created by a populator or some configuration utility like the JCL GUI editor.

When this method is successful, it initializes the properties **Claimed**, **DeviceEnabled**, **DataEventEnabled** and **FreezeEvents**, as well as descriptions and version numbers of the JavaPOS software layers. Additional category-specific properties may also be initialized.

Errors A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page B-12.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	The Control is already open.
JPOS_E_NOEXIST	The specified <i>logicalDeviceName</i> was not found.
JPOS_E_NOSERVICE	Could not establish a connection to the corresponding Device Service.

release Method

Syntax **void release () throws JposException;**

Remarks Releases exclusive access to the device.

If the **DeviceEnabled** property is true, and the device is an exclusive-use device, then the device is also disabled (this method does not change the device enabled state of sharable devices).

Errors A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page B-12.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	The application does not have exclusive access to the device.

resetStatistics Method**Added in Release 1.8****Syntax** **void resetStatistics (String *statisticsBuffer*) throws JposException;**

Parameter	Description
<i>statisticsBuffer</i>	The data buffer defining the statistics that are to be reset.

This is a comma-separated list of name(s), where an empty string (“”) means ALL resettable statistics are to be reset, “U_” means all UnifiedPOS defined resettable statistics are to be reset, “M_” means all manufacturer defined resettable statistics are to be reset, and “actual_name1, actual_name2” (from the XML file definitions) means that the specifically defined resettable statistic(s) are to be reset.

Remarks Resets the defined resettable statistics in a device.

Both **CapStatisticsReporting** and **CapUpdateStatistics** must be true in order to successfully use this method.

This method is always executed synchronously.

Errors A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page B-12.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	CapStatisticsReporting or CapUpdateStatistics is false, or the named statistic is not defined/resettable.

See Also **CapStatisticsReporting** Property, **CapUpdateStatistics** Property.**retrieveStatistics Method****Added in Release 1.8****Syntax** **void retrieveStatistics (String[1] *statisticsBuffer*) throws JposException;**

Parameter	Description
<i>statisticsBuffer</i>	The data buffer defining the statistics to be retrieved and in which the retrieved statistics are placed.

This is a comma-separated list of name(s), where an empty string (“”) means ALL statistics are to be retrieved, “U_” means all UnifiedPOS defined statistics are to be retrieved, “M_” means all manufacturer defined statistics are to be retrieved, and “actual_name1, actual_name2” (from the XML file definitions) means that the specifically defined statistic(s) are to be retrieved.

Remarks Retrieves the statistics from a device.

CapStatisticsReporting must be true in order to successfully use this method.

This method is always executed synchronously.

All calls to **retrieveStatistics** will return the following XML as a minimum:

```

<?xml version='1.0'?>
<UPOSStat version="1.8.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://www.nrf-arts.org/IXRetail/namespace/"
  xsi:schemaLocation=
    "http://www.nrf-arts.org/IXRetail/namespace/
    UPOSStat.xsd">
  <Event>
    <Parameter>
      <Name>RequestedStatistic</Name>
      <Value>1234</Value>
    </Parameter>
  </Event>
  <Equipment>
    <Manufacturer>
      <Name>Device Manufacturer</Name>
    </Manufacturer>
    <ModelNumber>Device Model Number</ModelNumber>
    <SensorID UPOS="POSPrinter"/>
  </Equipment>
</UPOSStat>

```

If the application requests a statistic name that the device does not support, the `<Parameter>` entry will be returned with an empty `<Value>`. e.g.,

```

<Parameter>
  <Name>RequestedStatistic</Name>
  <Value></Value>
</Parameter>

```

All statistics that the device collects that are manufacturer specific (not defined in the schema) will be returned in a `<ManufacturerSpecific>` tag instead of a `<Parameter>` tag. e.g.,

```

<ManufacturerSpecific>
  <Name>TheAnswer</Name>
  <Value>42</Value>
</ManufacturerSpecific>

```

When an application requests all statistics from the device, the device will return a `<Parameter>` entry for every defined statistic for the device category as defined by the XML schema version specified by the version attribute in the `<UPOSStat>` tag. If the device does not record any of the statistics, the `<Value>` tag will be empty.

The most up-to-date files defining the XML tag names and example schemas for the statistics for all device categories can be downloaded from the NRF-ARTS web site at <http://www.nrf-arts.org>.

Errors A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page B-12.

Some possible values of the exception’s `ErrorCode` property are:

Value	Meaning
JPOS_E_ILLEGAL	CapStatisticsReporting is false or the named statistic is not defined.

See Also **CapStatisticsReporting** Property.

updateStatistics Method**Added in Release 1.8****Syntax** **void updateStatistics (String *statisticsBuffer*) throws JposException;**

Parameter	Description
<i>statisticsBuffer</i>	The data buffer defining the statistics with values that are to be updated.

This is a comma-separated list of name-value pair(s), where an empty string name (“”=value1”) means ALL resettable statistics are to be set to the value “value1”, “U_=value2” means all UnifiedPOS defined resettable statistics are to be set to the value “value2”, “M_=value3” means all manufacturer defined resettable statistics are to be set to the value “value3”, and “actual_name1=value4, actual_name2=value5” (from the XML file definitions) means that the specifically defined resettable statistic(s) are to be set to the specified value(s).

Remarks Updates the defined resettable statistics in a device.

Both **CapStatisticsReporting** and **CapUpdateStatistics** must be true in order to successfully use this method.

This method is always executed synchronously.

Errors A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page B-12.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	CapStatisticsReporting or CapUpdateStatistics is false, or the named statistic is not defined/updatable.

See Also **CapStatisticsReporting** Property, **CapUpdateStatistics** Property.

Events

DataEvent

Interface `jpos.events.DataListener`

Method `dataOccurred (DataEvent e)`

Description Notifies the application that input data is available from the device.

Properties This event contains the following property:

Property	Type	Description
<i>Status</i>	<i>int</i>	The input status with its value dependent upon the device category; it may describe the type or qualities of the input data.

Remarks When this event is delivered to the application, the **DataEventEnabled** property is changed to false, so that no further data events will be delivered until the application sets **DataEventEnabled** back to true. The actual *byte array* input data is placed in one or more device-specific properties.

If **DataEventEnabled** is false at the time that data is received, then the data is enqueued in an internal buffer, the device-specific input data properties are not updated, and the event is not delivered. When **DataEventEnabled** is subsequently changed back to true, the event will be delivered immediately if input data is enqueued and **FreezeEvents** is false.

DirectIOEvent**Interface** `jpos.events.DirectIOListener`**Method** `directIOOccurred (DirectIOEvent e);`**Description** Provides Device Service information directly to the application. This event provides a means for a vendor-specific Device Service to provide events to the application that are not otherwise supported by the Device Control.**Properties** This event contains the following properties:

Property	Type	Description
<i>EventNumber</i>	<i>int</i>	Event number whose specific values are assigned by the Device Service.
<i>Data</i>	<i>int</i>	Additional numeric data. Specific values vary by the <i>EventNumber</i> and the Device Service. This property is settable.
<i>Object</i>	<i>Object</i>	Additional data whose usage varies by the <i>EventNumber</i> and the Device Service. This property is settable.

Remarks This event is to be used only for those types of vendor specific functions that are not otherwise described as part of the JavaPOS standard. Use of this event may restrict the application program from being used with other vendor's devices which may not have any knowledge of the Device Service's need for this event.

ErrorEvent**Updated in Release 1.7**

Interface	jpos.events.ErrorListener
Method	errorOccurred (ErrorEvent e);
Description	Notifies the application that an error has been detected and a suitable response is necessary to process the error condition.
Properties	This event contains the following properties:

Property	Type	Description
<i>ErrorCode</i>	<i>int</i>	Error Code causing the error event. See the list of <i>ErrorCodes</i> on page 16.
<i>ErrorCodeExtended</i>	<i>int</i>	Extended Error Code causing the error event. These values are device category specific.
<i>ErrorLocus</i>	<i>int</i>	Location of the error. See values below.
<i>ErrorResponse</i>	<i>int</i>	Error response, whose default value may be overridden by the application (i.e., this property is settable). See values below.

The *ErrorLocus* parameter has one of the following values:

Value	Meaning
JPOS_EL_OUTPUT	Error occurred while processing asynchronous output.
JPOS_EL_INPUT	Error occurred while gathering or processing event-driven input. No input data is available.
JPOS_EL_INPUT_DATA	Error occurred while gathering or processing event-driven input, and some previously buffered data is available.

The application's error event listener can set the *ErrorResponse* property to one of the following values:

Value	Meaning
JPOS_ER_RETRY	Retry the asynchronous output. The error state is exited. May be valid only when locus is JPOS_EL_INPUT. Default when locus is JPOS_EL_OUTPUT.
JPOS_ER_CLEAR	Clear all buffered output data (including all asynchronous output) or buffered input data. The error state is exited. Default when locus is JPOS_EL_INPUT.

JPOS_ER_CONTINUEINPUT

Acknowledges the error and directs the Device to continue input processing. The Device remains in the error state and will deliver additional **DataEvents** as directed by the **DataEventEnabled** property. When all input has been delivered and **DataEventEnabled** is again set to true, then another **ErrorEvent** is delivered with locus **JPOS_EL_INPUT**.

Use only when locus is **JPOS_EL_INPUT_DATA**.
Default when locus is **JPOS_EL_INPUT_DATA**.

Remarks This event is enqueued when an error is detected and the Device's **State** transitions into the error state. This event is not delivered until **DataEventEnabled** is true, so that proper application sequencing occurs.

OutputCompleteEvent

Interface **jpos.events.OutputCompleteListener**

Method **outputCompleteOccurred (OutputCompleteEvent e);**

Description Notifies the application that the queued output request associated with the *OutputID* property has completed successfully.

Properties This event contains the following property:

Property	Type	Description
<i>OutputID</i>	<i>int</i>	The ID number of the asynchronous output request that is complete.

Remarks This event is enqueued after the request's data has been both sent and the Device Service has confirmation that it was processed by the device successfully.

StatusUpdateEvent

Interface `jpos.events.StatusUpdateListener`

Method `statusUpdateOccurred (StatusUpdateEvent e);`

Description Notifies the application when a device has detected an operation status change.

Properties This event contains the following property:

Property	Type	Description
<i>Status</i>	<i>int</i>	Device category-specific status, describing the type of status change.

Note that Release 1.3 added Power State Reporting with additional *Status* values of:

Value	Meaning
JPOS_SUE_POWER_ONLINE	The device is powered on and ready for use. Can be returned if CapPowerReporting = JPOS_PR_STANDARD or JPOS_PR_ADVANCED.
JPOS_SUE_POWER_OFF	The device is off or detached from the terminal. Can only be returned if CapPowerReporting = JPOS_PR_ADVANCED.
JPOS_SUE_POWER_OFFLINE	The device is powered on but is either not ready or not able to respond to requests. Can only be returned if CapPowerReporting = JPOS_PR_ADVANCED.
POS_SUE_POWER_OFF_OFFLINE	The device is either off or offline. Can only be returned if CapPowerReporting = JPOS_PR_STANDARD. The common property PowerState is also maintained at the current power state of the device.

Remarks This event is enqueued when a Device needs to alert the application of a device status change. Examples are a change in the cash drawer position (open vs. closed) or a change in a POS printer sensor (form present vs. absent).

When a device is enabled, this event may be delivered to inform the application of the device state. This behavior, however, is not required.

Peripheral Interfaces

Note:

The following are two examples of how the proposed sections for each of the peripheral devices would be constructed. Where possible the tables are arranged to show the sequence of the commands for proper operation of the peripheral device.

The Cash Drawer and the MICR devices were chosen because they represent a simple output device and a more complex input device. The other peripheral devices would follow similar command usage and flow.

JavaPOS: Cash Drawer
Java Command Examples

OPERATION	T Y P E	JAVA SAMPLE	R E A D	W R I T E	A R G S	R T N V	E X C P	Ref Page
-----------	------------------	-------------	------------------	-----------------------	------------------	------------------	------------------	-------------

Initializing Properties, Methods, and Events

open *	M	myCashDrawer.open(LogicalDeviceName.CashDrawer);		•	1	void	•	52
claim *	M	myCashDrawer.claim(1000);		•	1	void	•	50
Claimed	P	bResult = myCashDrawer.getClaimed();	•			boolean	•	40
DeviceEnabled *	P	myCashDrawer.setDeviceEnabled(true);		•	1	-	•	42
DeviceEnabled	P	bResult = myCashDrawer.getDeviceEnabled();	•			boolean	•	42
DirectIO	M	myCashDrawer.directIO(100,int[],byte[])		•	3	void	•	51
CheckHealth	M	myCashDrawer.checkHealth(JPOS_CH_INTERNAL);		•	1	void	•	49
DirectIOEvent	E	public void directIOOccurred(DirectIOEvent e)			1	CMF		60

Capabilities, Assignments and Descriptions Properties, Methods, and Events

StatusUpdateEvent	E	public void statusUpdateOccurred(StatusUpdateEvent e)			1	CMF		63
CapPowerReporting	P	iResult = myCashDrawer.getCapPowerReporting();	•			int		38
CheckHealthText	P	sResult = myCashDrawer.getCheckHealthText();	•			String	•	38
FreezeEvents	P	myCashDrawer.setFreezeEvents(true);		•	1	-	•	44
FreezeEvents	P	bResult = myCashDrawer.getFreezeEvents();	•			boolean	•	44
PowerNotify	P	myCashDrawer.setPowerNotify(JPOS_PN_ENABLED);		•	1	-	•	45
PowerNotify	P	iResult = myCashDrawer.getPowerNotify();	•			int	•	45
PowerState	P	iResult = myCashDrawer.getPowerState();	•			int	•	46
PhysicalDevice Description	P	sResult = myCashDrawer.getPhysicalDeviceDescription();	•			String	•	47
PhysicalDevice Name	P	sResult = myCashDrawer.getPhysicalDeviceName();	•			String	•	47

OPERATION	T Y P E	JAVA SAMPLE	R E A D	W R I T E	A R G S	R T N V	E X C P	Ref Page
State	P	iResult = myCashDrawer.getState();	•			int		48
DeviceControl Description	P	sResult = myCashDrawer.getDeviceControlDescription();	•			String		41
DeviceControl Version	P	iResult = myCashDrawer.getDeviceControlVersion();	•			int		41
DeviceService Description	P	sResult = myCashDrawer.getDeviceServiceDescription();	•			String	•	42
DeviceService Version	P	iResult = myCashDrawer.getDeviceServiceVersion();	•			int	•	43

Cash Drawer Operations Properties, Methods, and Events

CapStatus	P	bResult = myCashDrawer.getCapStatus();	•			boolean	•	127
CapStatusMultiDrawerDetect	P	bResult = myCashDrawer.getCapStatusMultiDrawerDetect();	•			boolean	•	127
DrawerOpened	P	myCashDrawer.drawerOpened();	•			boolean	•	128
OpenDrawer *	M	myCashDrawer.openDrawer();		•		void	•	129
WaitForDrawerClose	M	myCashDrawer.waitForDrawerClose(2500, 1000, 10, 5);		•	4	void	•	129

Cash Drawer Terminating Methods

Release	M	myCashDrawer.release();		•		void	•	53
Close *	M	myCashDrawer.close();		•		void	•	51

Notes:

* Required for basic Cash Drawer operations

JavaPOS: MICR
Java Command Examples

OPERATION	T Y P E	JAVA SAMPLE	R E A D	W R I T E	A R G S	R T N V	E X C P	Ref Page
-----------	------------------	-------------	------------------	-----------------------	------------------	------------------	------------------	-------------

Initializing Properties, Methods, and Events

open *	M	myMicr.open(LogicalDeviceName.MICR);		•	1	void	•	52
claim *	M	myMicr.claim(1000);		•	1	void	•	50
Claimed	P	bResult = myMicr.getClaimed();	•			boolean	•	40
DeviceEnabled *	P	myMicr.setDeviceEnabled(true);		•	1	-	•	42
DeviceEnabled	P	bResult = myMicr.getDeviceEnabled();	•			boolean	•	42
AutoDisable	P	myMicr.setAutoDisable(true);		•	1	-	•	38
AutoDisable	P	bResult = myMicr.getAutoDisable();	•			boolean	•	38
DirectIO	M	myMicr.directIO(100,int[],byte[])		•	3	void	•	51
CheckHealth	M	myMicr.checkHealth(JPOS_CH_INTERNAL);		•	1	void	•	49
DirectIOEvent	E	public void directIOOccurred(DirectIOEvent e)			1	CMF		60
ErrorEvent	E	public void errorOccurred(ErrorEvent e)			1	CMF		61

Capabilities, Assignments and Descriptions Properties, Methods, and Events

StatusUpdateEvent	E	public void statusUpdateOccurred(StatusUpdateEvent e)			1	CMF		63
CapPowerReporting	P	iResult = myMicr.getCapPowerReporting();	•			int		38
CheckHealthText	P	sResult = myMicr.getCheckHealthText();	•			String	•	38
DataCount	P	iResult = myMicr.getDataCount();	•			int	•	40
FreezeEvents	P	myMicr.setFreezeEvents(true);		•	1	-	•	44
FreezeEvents	P	bResult = myMicr.getFreezeEvents();	•			boolean	•	44
PowerNotify	P	myMicr.setPowerNotify(JPOS_PN_ENABLED);		•	1	-	•	45
PowerNotify	P	iResult = myMicr.getPowerNotify();	•			int	•	45

OPERATION	T Y P E	JAVA SAMPLE	R E A D	W R I T E	A R G S	R T N V	E X C P	Ref Page
PowerState	P	iResult = myMicr.getPowerState();	•			int	•	46
PhysicalDevice Description	P	sResult = myMicr.getPhysicalDeviceDescription();	•			String	•	47
PhysicalDevice Name	P	sResult = myMicr.getPhysicalDeviceName();	•			String	•	47
State	P	iResult = myMicr.getState();	•			int		48
DeviceControl Description	P	sResult = myMicr.getDeviceControlDescription();	•			String		41
DeviceControl Version	P	iResult = myMicr.getDeviceControlVersion();	•			int		41
DeviceService Description	P	sResult = myMicr.getDeviceServiceDescription();	•			String	•	42
DeviceService Version	P	iResult = myMicr.getDeviceServiceVersion();	•			int	•	43

MICR Operations Properties, Methods, and Events

OPERATION	T Y P E	JAVA SAMPLE	R E A D	W R I T E	A R G S	R T N V	E X C P	Ref Page
CapValidationDevice	P	bResult = myMicr.getCapValidationDevice();	•			boolean	•	461
ClearInput	M	myMicr.clearInput();		•		void	•	50
DataEventEnabled *	P	myMicr.setDataEventEnabled(true);		•	1	-	•	40
DataEventEnabled	P	bResult = myMicr.getDataEventEnabled();	•			boolean	•	40
BeginInsertion *	M	myMicr.beginInsertion(2000);		•	1	void	•	464
EndInsertion *	M	myMicr.endInsertion();		•		void	•	466
DataEvent	E	public void dataOccurred(DataEvent e)			1	CMF		59
BeginRemoval *	M	myMicr.beginRemoval(1000);		•		void	•	465
EndRemoval *	M	myMicr.endRemoval();		•		void	•	467
RawData	P	sResult = myMicr.getRawData();	•			String	•	463
AccountNumber	P	sResult = myMicr.getAccountNumber();	•			String	•	460
Amount	P	sResult = myMicr.getAmount();	•			String	•	460
BankNumber	P	sResult = myMicr.getBankNumber();	•			String	•	460
EPC	P	sResult = myMicr.getEPC();	•			String	•	462
SerialNumber	P	sResult = myMicr.getSerialNumber();	•			String	•	463
TransitNumber	P	sResult = myMicr.getTransitNumber();	•			String	•	463
CheckType	P	iResult = myMicr.getCheckType();	•			int	•	461
CountryCode	P	iResult = myMicr.getCountryCode();	•			int	•	462

MICR Terminating Methods

Release	M	myMicr.release();		•		void	•	53
Close *	M	myMicr.close();		•		void	•	51

* Required for basic MICR operations

Section 3: Technical Details - OPOS and JavaPOS

The Java for Retail POS (JavaPOS) and OLE for Retail POS (OPOS) industry standard initiatives are intentionally similar in many respects since the UnifiedPOS architecture is the basis from which JavaPOS and OPOS implementations are derived. The most up to date information can be downloaded from the web site, <http://www.nrf-arts.org>, under the JavaPOS Standard files section.

Support for Java requires several differences from OPOS in architecture, but the JavaPOS committee agreed that the general model of OPOS device classes should be reused as much as possible.

In order to reuse as much of the OPOS device models as possible, the following sections detail the general mapping rules from OPOS to JavaPOS. A later section lists the deviations of JavaPOS APIs from OPOS.

OPOS to JavaPOS - API Mapping Rules

In most cases, OPOS APIs may be translated in a mechanical fashion to equivalent JavaPOS APIs. The exceptions to this mapping are largely due to differences in some string parameters.

Areas of data mapping include data types, methods and properties, and events.

Data Types

Data types are mapped from OPOS to JavaPOS as follows, with exceptions noted after the table:

Table 2:

OPOS Type	JavaPOS Type	Usage
BOOL	boolean	Boolean true or false.
BOOL *	boolean[1]	Modifiable boolean.
LONG	int	32-bit integer.
LONG *	int[1]	Modifiable 32-bit integer.
CURRENCY	long	64-bit integer. Used for currency values, with an assumed 4 decimal places.
CURRENCY *	long[1]	Modifiable 64-bit integer.
		<i>The string types are usually represented with the following mapping:</i>
BSTR	String	Text character string.
BSTR *	String[1]	Modifiable text character string.
		<i>For some APIs, the string types are represented in one of the following:</i>
	byte[]	Array of bytes. May be modified, but size of array cannot be changed. Often used when non-textual data is possible.
	Point[]	Array of points. Used by Signature Capture.
	Object	An object. This will usually be subclassed to provide a Device Service-specific parameter for directIO or DirectIOEvent .
nls (LONG)	nls (String)	Operating System National Language Data type.

Property and Method Names

Property and method names are mapped from OPOS to JavaPOS as follows:

Table 3:

Type	OPOS Examples	JavaPOS Examples	Mapping Rule
Property Read	Claimed DeviceEnabled OutputID	getClaimed() getDeviceEnabled() getOutputID()	Prepend “get” to the property name to form the property accessor method. No parameters. Return value is the property.
Property Write	AutoDisable DeviceEnabled	setAutoDisable(...) setDeviceEnabled(...)	Prepend “set” to the property name to form the property mutator method. One parameter, which is of the property's type. No return value.
Method	Open CheckHealth DirectIO	open checkHealth directIO	Change first letter to lowercase. Other characters are unchanged.

Events

JavaPOS events use the Java Development Kit 1.1 event delegation model, whereby the application registers for events, supplying a class instance that implements an interface extended from **EventListener**.

For each *Event* type which the Application wishes to receive, the Application must implement the corresponding **jpos.events.EventListener** interface and handle its event method. Events are delivered by the JavaPOS Device by calling this event method.

Constants

Constants are mapped from OPOS to JavaPOS as follows:

- If the constant begins with “OPOS”, then change “OPOS” to “JPOS.”
- Otherwise, make no changes to the constant name.

All constant interface files are available in the package “jpos.” All constants are of type “static final int.”

API Deviations

The following OPOS APIs do not follow the above mapping rules:

- **BinaryConversion** property
Not needed by JavaPOS. This OPOS property was used to overcome a COM-specific issue with passing binary data in strings. JavaPOS uses more appropriate types for these cases, such as byte arrays.
- **OpenResult** property
Not supported by JavaPOS.
- **ResultCode** and **ResultCodeExtended** properties
Not needed by JavaPOS. These OPOS properties are used for reporting failures on method calls and property sets. In JavaPOS, these failures (plus property get failures) cause a **JposException**. This exception includes the properties **ErrorCode** and **ErrorCodeExtended**, with values that match the OPOS properties.
- **ClaimDevice** method
In OPOS, this method was introduced in Release 1.5. Previous releases defined the **Claim** method. This method is **claim** in all releases of JavaPOS.
- **ReleaseDevice** method
In OPOS, this method was introduced in Release 1.5. Previous releases defined the **Release** method. This method is **release** in all releases of JavaPOS.
- **DirectIO** method and **DirectIOEvent**
The BSTR* parameter is mapped to Object.
- Cash Drawer **WaitForDrawerClosed** method
The tone function of this method may not work on non-PCs, since it depends on the availability of a speaker.
- Hard Totals **Read** method
The BSTR* parameter is mapped to byte[], with its size set to the requested number of bytes.
- Hard Totals **Write** method
The BSTR parameter is mapped to byte[].
- MSR **Track1Data**, **Track1DiscretionaryData**, **Track2Data**, **Track2DiscretionaryData**, **Track3Data** properties
These BSTR properties are mapped to byte[].
- PINPad **PromptLanguage** property
This LONG property is mapped to String.
- Scanner **ScanData** and **ScanDataLabel** properties
These BSTR properties are mapped to byte[].
- Signature Capture **PointArray** property
This BSTR property is mapped to Point[].
- Signature Capture **RawData** property
This BSTR property is mapped to byte[].
- Signature Capture **TotalPoints** property
Not needed by JavaPOS. This property is equivalent to “**PointArray.length**”, so **TotalPoints** is redundant.

Mapping of CharacterSet

Added in Release 1.7

For some devices like the POSPrinter, LineDisplay, ROD, and PCRW, it is necessary to select an appropriate character set in the **CharacterSet** property of the Service when printing or displaying characters on the device. Usually a JavaPOS application uses the unicode character set. When the device only contains three-digit code pages - such as 850 or 852 - a mapping of the characters from their positions in the application-side unicode character set to the device-side code page is necessary.

The following code snippet allows Device Service providers to easily add the mapping mechanism into their Services. For mapping of the characters, the encoding capabilities of the Java Runtime Environment (JRE) are used. (It is assumed that the data transferred to the Service for output to the device is a String, and that the lower software layers, such as comm.api, use byte arrays.)

```
/** converts a string with the appropriate code page to a byte array.
 * @param codePage the desired code page to which
 *         the characters should be mapped - such as 1252 or 850...
 * @param src the source string to be mapped.
 * @return the mapped character as byte array.
 *         Returns null if mapping to this codepage is not supported.
 */
static byte[] UnicodeToOEMCodePage (int codePage, String src)
{
    try { return src.getBytes ("Cp" + codePage);}
    catch (java.io.UnsupportedEncodingException e) {}
    return null;
}
```

Note:

- The used (extended) encoding set of the Java Runtime Environment must be installed. Usually, the i18n package is required.
- Refer to the Java SDK documentation for the term *Internationalization*.

Section 4: JavaPOS Change History

Release 1.3

Release 1.3 adds additional device classes, a few additional APIs, and some corrections. Release 1.3 is a superset of Release 1.2.

<u>Section</u>	<u>Change</u>
General	Modify the use of the term event “firing.” Use “enqueue” and “deliver” appropriately to describe event firing.
Bump Bar	New device: Add information in several locations, plus Bump Bar chapter and interface files.
Fiscal Printer	New device: Add information in several locations, plus Fiscal Printer chapter and interface files.
PIN Pad	New device: Add information in several locations, plus PIN Pad chapter and interface files.
Remote Order Display	New device: Add information in several locations, plus Remote Order Display chapter and interface files.
Several places	Relax ErrorEvent “retry” response to allow its use with some input devices.
Introduction Events	Clarify effect of the top event being blocked.
Introduction Input Model	Add details concerning enqueueing and delivering ErrorEvents . Add description of asynchronous input.
Introduction Device Power Reporting Model	Add this section.
Common CapPowerReporting , PowerNotify , PowerState properties	Add these sections.
Common ErrorCode property	Generalize the meaning of JPOS_E_BUSY.
Common StatusUpdateEvent	Add power state reporting information. Change parameter name from <i>Data</i> to <i>Status</i> .
Every Device	Add power reporting properties to Summary section. Add StatusUpdateEvent support (if previously not reported). Add power reporting reference to existing StatusUpdateEvent descriptions.
MSR DecodeData	Add “raw format” description and column to track data table.
MSR ExpirationDate	Specify the format.

MSR TrackData	Specify that data excludes the sentinels and LRC. Add that decoding occurs when DecodeData is true.
MSR ErrorEvent	Clarify that DataCount and AutoDisable are not relevant for MSR error events.
POSPrinter XxxLineChars	Add implementation recommendations.
POSPrinter printTwoNormal	Clarify the meaning of the <i>stations</i> parameter, including the addition of new constants.
Scale	Add the following features: <ul style="list-style-type: none"> • Asynchronous input. Property AsyncMode. Method clearInput, updates to readWeight. Events DataEvent and ErrorEvent. • Display of text. Properties CapDisplayText, MaxDisplayTextChars. Method displayText. • Price calculation. Properties CapPriceCalculating, SalesPrice, UnitPrice. • Tare weight. Properties CapTareWeight, TareWeight. • Scale zeroing. Property CapZeroScale. Method zeroScale.
Tone Indicator Summary and General Information's Device Sharing	Consistently specify that Tone Indicator is a sharable device.
JposConst.java interface files	Add CapPowerReporting , PowerState , and PowerNotify properties. Add StatusUpdateEvent power reporting values.
POSPrinterConst.java interface files	Add new printTwoNormal station constants.
Throughout	Correct some editing errors.

Release 1.4

Release 1.4 added the additional peripheral device, Credit Authorization Terminal (CAT). This device, as specified, is currently only used in the Japanese POS markets.

Addition of this device required re-ordering the chapters and modifications to the Table of Contents. Other minor changes to the standard are as noted below.

Release 1.4 is a superset of Release 1.3.

<u>Section</u>	<u>Change</u>
General	Update the Package Structure on page B-37 to include CAT device; update the files to correct some erroneous references to OPOS.

Fiscal Printer	Add clarification to when the ErrorStation property is valid.
POS Printer	Add clarification to when the ErrorStation property is valid.
Appendix B	Add clarification to the “Events” section description.
Throughout	Correct interface name to jpos.events.OutputCompleteListener . Correct minor spelling errors.

Release 1.5

Release 1.5 adds two additional peripheral devices: Pointcard Reader Writer and POSPower, incorporates additional clarifications to the standard, adds a few new additional APIs for some of the existing devices, and makes some corrections to insure consistency in the device descriptions. Release 1.5 is a superset of Release 1.4.

<u>Section</u>	<u>Change</u>
Throughout	Correct notation for Java Unicode to “\uxxxx”
General	Add clarification to when the Device exits the Error state. Remove the JPS documentation from the standard. The JPS implementation has been replaced with the JCL mechanism for locating and maintaining the Java Device Services. Revised the tables and diagrams as necessary to reflect these changes. Update the Standard and the Package Structure to reflect the additional new devices added to this version.
Common Properties, Methods, and Events	Modified General section to reflect JDK version dependencies.
Bump Bar	Add clarification that this Device can be both an input and an output device.
Cash Changer	Add the necessary properties (DataCount , DataEventEnabled , CapDeposit , CapDepositDataEvent , CapPauseDeposit , CapRepayDeposit , DepositAmount , DepositCashList , DepositCodeList , DepositCounts , DepositStatus), methods (beginDeposit , endDeposit , fixDeposit , pauseDeposit) and events (DataEvent) for this device to optionally be able to handle cash acceptance.
Cash Drawer	Added new property, CapStatusMultiDrawerDetect to improve status reporting in multiple cash drawer environments.
CAT	Correct the properties section to reflect the correct data

	type for TransactionType (an integer) and TransactionNumber (a String); other minor corrections to fix typographical errors.
Coin Dispenser	No Changes
Fiscal Printer	<p>Added Russia to list of countries in the CountryCode property.</p> <p>Added note to clarify that Currency value is specified to be four decimal places.</p> <p>Changed the properties CountryCode, ErrorOutID, PrinterState, QuantityDecimalPlaces, and QuantityLength to clarify when the parameters are Initialized.</p> <p>Corrected DuplicateReceipt to show that it is a R/W Property.</p>
Hard Totals	No Changes
Keylock	No Changes
Line Display	Clarify properties CharacterSet and CharacterSetList to indicate when they are initialized and to what values they may be set.
MICR	<p>Added clarification to description of Model concerning the availability of parsed data.</p> <p>Clarify number of digits for BankNumber as specified by ABA Standard, Thomson Financial Publishing Inc.</p>
MSR	<p>Added properties CapTransmitSentinels, Track4Data, and TransmitSentinels to enhance the features that may be available in a global MSR device.</p> <p>Updated the status byte definitions for the DataEvent event.</p>
Pin Pad	<p>Added the Track4Data property.</p> <p>Clarify that Track1Data, Track2Data, Track3Data, and Track4Data are assumed to be decoded data if a successful read takes place.</p>
Pointcard Reader Writer	New device classification added to the standard. This device is used primarily in Asian markets.
POS Keyboard	CapKeyUp property type corrected from Long to boolean
POS Power	New device classification added to the standard to allow for systems that have the capability to report and manage alternative mains power (UPS type devices).
POS Printer	Revise this device classification to include properties, methods, and events to add multi-color printing, both side printing for documents such as checks, and marked

	paper and sensing capability for special POS printer forms handling. This section had significant changes to the General Information section as well to help clarify standard to reduce the possibility of creating a Device Service that does not meet the intent of the standard.
ROD	Clarify model remarks to indicate that this device can be both an output device and an input device. Clarify General Model description explaining how Applications can manage and control the Remote Order Displays. Clarify to indicate that ErrorUnits and ErrorString are updated instead by <u>synchronous</u> broadcast method. Clarify what value the CurrentUnitID property is initialized.
Scale	Clarify the properties SalesPrice , TareWeight , and UnitPrice to indicate when the values are initialized and can be expected to remain stable and valid.
Scanner (Bar Code Reader)	No Changes
Signature Capture	Update Model to discuss AutoDisable implications; clarify when RealTimeDataEnabled takes effect; correct DataEvent to indicate when this event may be fired to include real-time data.
Tone Indicator	Clarify all the specific properties to indicate when the values are initialized and can be expected to remain stable and valid. Also clarify handling of the Sound method when another application claims the device and calls the Sound method.

Release 1.6

Release 1.6 does not add any new devices to the standard but does make significant changes to the Fiscal Printer and Line Display devices. Additional minor clarification and correction changes are added as noted below. Release 1.6 is a superset of Release 1.5.

<u>Section</u>	<u>Change</u>
Fiscal Printer	Added the CapAdditionalHeader , CapAdditionalTrailer , CapChangeDue , CapEmptyReceiptIsVoidable , CapFiscalReceiptStation , CapFiscalReceiptType , CapMultiContractor , CapOnlyVoidLastItem , CapPackageAdjustment , CapPostPreLine , CapSetCurrency , CapTotalizerType , ActualCurrency , AdditionHeader , AdditionalTrailer , ChangeDue , ContractorId ,

	<p>DateType, FiscalReceiptStation, FiscalReceiptType, MessageType, PostLine, PreLine, and TotalizerType properties.</p> <p>Added the setCurrency, printRecCash, printRecItemFuel, printRecItemFuelVoid, printRecPackageAdjustment, printRecPackageAdjustVoid, printRecRefundVoid, printRecSubtotalAdjustVoid, and printRecTaxID methods.</p> <p>Clarified the description of the CapPositiveAdjustment property.</p> <p>Added country support for Bulgaria and Romania.</p> <p>Updated the CountryCode, DayOpened, and DescriptionLength properties to reflect additions to the specification.</p> <p>Updated the endFiscalReceipt, getData, getDate, printRecItem, printRecMessage, printRecNotPaid, printRecRefund, printRecSubtotal, printRecSubtotalAdjustment, printRecTotal, printRecVoid, printRecVoidItem, printZReport, and setHeaderLine methods to reflect additions to the specification.</p> <p>Updated ErrorEvent to reflect additions to the specification.</p> <p>Properties CountryCode, ErrorOutputID, PrinterState, QuantityDecimalPlaces, and QuantityLength have been updated to reflect the fact that they should be initialized after open instead of open, claim, and enable.</p> <p>Many updates in the General Information section.</p>
Line Display	<p>Added CapBlinkRate, CapCursorType, CapCustomGlyph, CapReadBack, CapReverse, BlinkRate, CursorType, CustomGlyphList, GlyphHeight, and GlyphWidth properties.</p> <p>Added defineGlyph and readCharacterAtCursor methods.</p> <p>Updated the displayText and displayTextAt methods to support new attributes for reverse video, DISP_DT_REVERSE and DISP_DT_BLINK_REVERSE.</p>
Scale	<p>Properties SalesPrice, TareWeight, and UnitPrice have been updated when the parameters are initialized following an open method.</p>
Tone Indicator	<p>Properties AsyncMode, Tone1Pitch, Tone1Volume,</p>

Tone1Duration, Tone2Pitch, Tone2Volume, Tone2Duration, and InterToneWait have been updated to reflect the fact that they should be initialized after **open** instead of **open, claim, and enable**.

Clarified handling of the **sound** method when another application claims the device and calls the **sound** method.

Release 1.7

The change history above has been maintained to this point for historical reference.

No specific change history relative to the JavaPOS Programming Guide is maintained from this release forward. Refer to Appendix C for the change history details (if any) relative to this section.

Change History

Release Version 1.4

Version 1.4 is the first release of the UnifiedPOS standard, and was issued on February 25, 1999. 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, issued on September 24, 2000, 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.
- 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 7.
- Updated the Package Diagram. See “Package Diagram” on page 8.
- 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 20.
- Updated the Power State Diagram. See “Power State Diagram” on page 23.
- Updated the Device State Diagram. See “Device State Diagram” on page 30.
- 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 37.
- Updated the Event Interfaces Diagram. See “upos::events interfaces” on page 58.
- Updated the Bump Bar chapter header to remove the “example” status. See “CHAPTER 2 BUMP BAR” on page 65.
- Updated the Bump Bar Class Diagram. See “Bump Bar Class Diagram” on page 70.
- Updated the Bump Bar State Diagram. See “Bump Bar State Diagram” on page 74.
- Added a new chapter describing the Cash Changer, including 1.5 specific updates. See “CHAPTER 3 Cash Changer” on page 91.
- Added a new chapter describing the Cash Drawer, including 1.5 specific updates. See “CHAPTER 4 CASH DRAWER” on page 121.
- Added a new chapter describing the CAT, including 1.5 specific updates. See “CHAPTER 5 CAT - Credit Authorization Terminal” on page 133.
- Added a new chapter describing the MSR. See “CHAPTER 14 MSR - MAGNETIC STRIPE READER” on page 483.
- 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 Sequence Diagram” on page 489.
- Added a new chapter describing the PIN Pad, including 1.5 specific updates. See “CHAPTER 15 PIN Pad” on page 509.
- Added a new chapter describing the Point Card Reader Writer. See “CHAPTER 16 POINT CARD READER WRITER” on page 539.
- Added a new chapter describing the POS Power. See “CHAPTER 18 POS Power” on page 601.
- Added a new chapter describing the POS Printer. See “CHAPTER 19 POS Printer” on page 621.
- 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 637. See “CapSlpBothSidesPrint Property Added in Release 1.5” on page 660. See “changePrintSide Method Added in Release 1.5” on page 693.
- Added a new Appendix describing Hardware References. See “APPENDIX E Additional Hardware References” on page E-1.
- Made minor typographical and formatting changes as necessary.

Release Version 1.6

Version 1.6 of this specification, issued on July 15, 2001, contains several new/completed chapters (not new devices) and updates to existing chapters that provide updates, clarifications, and corrections to Version 1.5. 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.
- Updated the Table of Contents to reflect additional chapters and headings. “TABLE OF CONTENTS” on page i
- Completed the chapter describing the Coin Dispenser device. See “CHAPTER 7 COIN DISPENSER” on page 213.
- Completed the chapter describing the Fiscal Printer device. See “CHAPTER 8 FISCAL PRINTER” on page 225.
 - Added the **CapAdditionalHeader**, **CapAdditionalTrailer**, **CapChangeDue**, **CapEmptyReceiptIsVoidable**, **CapFiscalReceiptStation**, **CapFiscalReceiptType**, **CapMultiContractor**, **CapOnlyVoidLastItem**, **CapPackageAdjustment**, **CapPostPreLine**, **CapSetCurrency**, **CapTotalizerType**, **ActualCurrency**, **AdditionHeader**, **AdditionalTrailer**, **ChangeDue**, **ContractorId**, **DateType**, **FiscalReceiptStation**, **FiscalReceiptType**, **MessageType**, **PostLine**, **PreLine**, and **TotalizerType** properties.
 - Changed the descriptions of the following properties to indicate that initialization takes place when the device is first enabled following the **open** method call: **CountryCode**, **ErrorOutID**, **PrinterState**, **QuantityDecimalPlaces**, and **QuantityLength**.
 - Added the **setCurrency**, **printRecCash**, **printRecItemFuel**, **printRecItemFuelVoid**, **printRecPackageAdjustment**, **printRecPackageAdjustVoid**, **printRecRefundVoid**, **printRecSubtotalAdjustVoid**, and **printRecTaxID** methods.
 - Added country support for Bulgaria and Romania.
 - Many updates in the **General Information** section.
 - Clarified the description of the **CapPositiveAdjustment** property.
 - Updated the **CountryCode**, **DayOpened**, and **DescriptionLength** properties to reflect additions to the specification.
 - Updated the **endFiscalReceipt**, **getData**, **getDate**, **printRecItem**, **printRecMessage**, **printRecNotPaid**, **printRecRefund**, **printRecSubtotal**, **printRecSubtotalAdjustment**, **printRecTotal**, **printRecVoid**, **printRecVoidItem**, **printZReport**, and **setHeaderLine** methods to reflect additions to the specification.
 - Updated **ErrorEvent** to reflect additions to the specification.
- Completed the chapter describing the Hard Totals device. See “CHAPTER 9 HARD TOTALS” on page 369.

- Completed the chapter describing the Keylock device. See “CHAPTER 10 KEYLOCK” on page 393.
- Completed the chapter describing the Line Display device. See “CHAPTER 11 LINE DISPLAY” on page 403.
 - Added **CapBlinkRate**, **CapCursorType**, **CapCustomGlyph**, **CapReadBack**, **CapReverse**, **BlinkRate**, **CursorType**, **CustomGlyphList**, **GlyphHeight**, and **GlyphWidth** properties.
 - Added **defineGlyph** and **readCharacterAtCursor** methods.
 - Updated the **displayText** and **displayTextAt** methods to support new attributes for reverse video, **DISP_DT_REVERSE** and **DISP_DT_BLINK_REVERSE**.
- Completed the chapter describing the MICR device. See “CHAPTER 12 MICR - MAGNETIC INK CHARACTER RECOGNITION READER” on page 451.
- Completed the chapter describing the POS Keyboard device. See “CHAPTER 17 POS KEYBOARD” on page 589.
- Completed the chapter describing the Remote Operator Display device. See “CHAPTER 20 REMOTE ORDER DISPLAY” on page 729.
- Completed the chapter describing the Scale device. See “CHAPTER 21 SCALE” on page 773.
 - Changed the descriptions of the following properties to indicate that initialization takes place when the device is first enabled following the **open** method call:
SalesPrice, **TareWeight**, and **UnitPrice**.
- Completed the chapter describing the Scanner device. See “CHAPTER 22 SCANNER (BAR CODE READER)” on page 791.
- Completed the chapter describing the Signature Capture device. See “CHAPTER 23 SIGNATURE CAPTURE” on page 805.
- Completed the chapter describing the Tone Indicator device. See “CHAPTER 25 TONE INDICATOR” on page 853.
 - Changed the descriptions of the following properties to indicate that initialization takes place when the device is first enabled following the **open** method call:
AsyncMode, **InterToneWait**, **Tone1Duration**, **Tone1Pitch**, **Tone1Volume**, **Tone2Duration**, **Tone2Pitch**, and **Tone2Volume**.
- Reformatted the Tables in the Summary sections of each chapter and included the original version in which the Properties, Methods, and Events were supported.
- Moved Appendices A, B, and C to be Appendices C, D, and E to make room for the OPOS and JavaPOS Appendices. See “APPENDIX C Change History” on page C-1, “APPENDIX D Additional Software References” on page D-1, and also “APPENDIX E Additional Hardware References” on page E-1.

Release Version 1.7

Version 1.7 of this specification, released on July 24, 2002, includes chapters describing two new devices, Check Scanner and Motion Sensor, and contains several updates to the existing chapters that provide enhancements, clarifications, and corrections to Version 1.6. These changes are detailed below, with links to the corresponding pages and/or chapters as appropriate. However, any minor typographical changes are not listed below.

- Updated the Version and issue date on the front page.
- Added the NRF Copyright notice. See page ii.
- Added the NRF Disclaimer notice. See page ii.
- Updated the Table of Contents to reflect additional sections. See “TABLE OF CONTENTS” on page i.
- Expanded the wording in several chapters to clarify the meaning of “Buffers the request.” to be “Buffers the request in program memory, for delivery to the Physical Device as soon as the Physical Device can receive and process it.”, or similar wording. The following chapters incorporate this change:
 - Introduction and Architecture
 - Bump Bar
 - Fiscal Printer
 - Point Card Reader/Writer
 - POS Printer
 - Remote Order Display
 - Tone Indicator
 - Appendix A - OPOS
 - Appendix B - JavaPOS
- Expanded/clarified the definition in several chapters of the `ER_CLEAR` *ErrorResponse* to an **ErrorEvent**. The following chapters incorporate this change:
 - Common Properties, Methods, and Events
 - Bump Bar
 - Fiscal Printer
 - Point Card Reader/Writer
 - POS Printer
 - Remote Order Display
 - Tone Indicator
 - Appendix A - OPOS (also **SOError**)
 - Appendix B - JavaPOS
- Expanded/clarified the definition in several chapters of the function of the **clearOutput** method. The following chapters incorporate this change:
 - Common Properties, Methods, and Events
 - Bump Bar
 - Remote Order Display
 - Appendix A - OPOS
 - Appendix B - JavaPOS
- Used a consistent description of “**XXXXXXEvent** being delivered to the application” in the following chapters:
 - MICR, Scanner, and SignatureCapture devices.

- Reworded the Dependencies section to reference Appendices A and B as the implementation reference, see page 3.
- Reworded the application's requirements for Event registration, see page 10.
- Added OPOS and JavaPOS verbiage, listed the OPOS-specific Common Property names, and cross reference links to the language specific Common Properties Summary Tables from the Common Properties Summary Table, see page 33.
- Added clarification of the initial value of the **PowerNotify** property after the **open** method call, see "PowerNotify Property" on page 45.
- Added a sequence diagram to the **open** method description. See page 52.
- Updated the Common **DirectIOEvent** *Obj* attribute to reference the OPOS **BinaryConversion** property, see page 60.
- Expanded the meaning of the ER_RETRY *ErrorResponse* attribute of the **ErrorEvent**, see page 61.
- Corrected the values for **ErrorEvent** *ErrorLocus* and *ErrorResponse* attributes from E_EL_XXX and E_ER_XXX to EL_XXX and ER_XXX, see page 61.
- Added a Sequence Diagram to the Cash Changer device chapter, see page 100. This diagram replaces the "processing flow" diagram.
- Added a Sequence Diagram to the Cash Drawer device chapter, see page 125.
- Changed the chapter heading for CAT to be "CAT - Credit Authorization Terminal" for consistency.
- Added a Sequence Diagram to the CAT device chapter, see page 144.
- Updated the CAT property **AdditionalSecurityInformation** to reference the OPOS **BinaryConversion** property, see page 146.
- Updated the CAT property **SlipNumber** to be consistently defined as a string in the Summary and Properties section of the chapter, see page 161.
- Reworded some of the descriptions in the CAT, **ErrorEvent**, Attributes section, see page 171.
- Added the chapter describing the Check Scanner device. See "CHAPTER 6 CHECK SCANNER" on page 175. The chapters following have been renumbered accordingly.
- Added a Sequence Diagram to the CoinDispenser device chapter, see page 218.
- Removed two blank (headings only) pages from the FiscalPrinter chapter that were to contain diagrams, namely, the Fiscal Printer State Diagram and the Fiscal Printer PrinterState Diagram.
- Updated the FiscalPrinter **printNormal** method *data* parameter to reference the OPOS **BinaryConversion** property, see page 314.
- Added a Sequence Diagram to the HardTotals device chapter, see page 375.
- Corrected the *ErrorCode* value for **commitTrans** to E_ILLEGAL, see page 382.
- Updated the HardTotals **read** method *data* parameter to reference the OPOS **BinaryConversion** property, see page 386.

- Added the *ErrorCode* value of E_ILLEGAL to the **setAll** method, see page 389.
- Updated the HardTotals **write** method *data* parameter to reference the OPOS **BinaryConversion** property, see page 390.
- Updated/corrected the Class Diagram of the Keylock device chapter, see page 396.
- Added a Sequence Diagram to the Keylock device chapter, see page 397.
- Deleted the last (redundant) bullet of the Capabilities section in the LineDisplay device chapter, see page 407.
- Updated the Class Diagram of the LineDisplay device chapter, see page 408.
- Added a Sequence Diagram to the LineDisplay device chapter, see page 409.
- Added a Data Characters and Escape Sequence section to the LineDisplay device chapter, see page 412.
- Updated the LineDisplay **DeviceColumns** property to reflect the impact of changing **ScreenMode**, see page 423.
- Updated the LineDisplay **DeviceRows** property to reflect the impact of changing **ScreenMode**, see page 423.
- Updated the LineDisplay device to support CodePage mapping:
 - Added the following properties: **CapMapCharacterSet** and **MapCharacterSet**.
- Updated the LineDisplay device to support various screen modes:
 - Added the following properties: **CapScreenMode**, **ScreenMode**, and **ScreenModeList**.
- Updated the LineDisplay device to support the displaying of bitmaps:
 - Added the following properties: **CapBitmap**, **MaximumX**, and **MaximumY**.
 - Added the following methods: **displayBitmap**, **setBitmap**.
- Updated the LineDisplay **clearText** method to clarify the lifetime of bitmaps, see page 434.
- Updated the LineDisplay **defineGlyph** method *glyph* parameter to reference the OPOS **BinaryConversion** property, see page 436.
- Updated the LineDisplay **displayText** method *data* parameter to reference the OPOS **BinaryConversion** property, see page 440.
- Updated the LineDisplay **displayText** method to reference the use of escape sequences and the placement of text and bitmaps, see page 440.
- Updated the LineDisplay **displayTextAt** method *data* parameter to reference the OPOS **BinaryConversion** property, see page 442.
- Updated the LineDisplay **scrollText** method to clarify that bitmaps are also scrolled, see page 444.
- Changed the chapter heading for MICR to be “MICR - Magnetic Ink Character Recognition Reader” for consistency.
- Added a Sequence Diagram to the MICR device chapter, see page 456.
- Expanded the description of the check removal processing under the Model section, see page 458.

- Expanded the description of event firing after the **endInsertion** processing is successfully completed, see page 466.
- Added additional *ErrorCodeExtended* values to the MICR **ErrorEvent**, see page 469.
- Added the chapter describing the Motion Sensor device. See “CHAPTER 13 MOTION SENSOR” on page 473. The chapters following have been renumbered accordingly.
- Changed the chapter heading for MSR to be “MSR - Magnetic Stripe Reader” for consistency.
- Added a Sequence Diagram to the MSR device chapter, see page 489.
- Added a Sequence Diagram to the PINPad device chapter, see page 515.
- Updated the PINPad **computeMAC** method *inMsg* and *outMsg* parameters to reference the OPOS **BinaryConversion** property, see page 531.
- Added a new ESC sequence to the Point Card Reader Writer device chapter providing for more reliable handling of pass through data, see page 551.
- Added a Sequence Diagram to the Point Card Reader Writer device chapter, see page 553.
- Updated the Point Card Reader Writer device to support CodePage mapping by adding the **CapMapCharacterSet** (see page 557) and **MapCharacterSet** (see page 564) properties.
- Updated the Point Card Reader Writer **printWrite** method *data* parameter to reference the OPOS **BinaryConversion** property, see page 581.
- Updated the Point Card Reader Writer **validateData** method *data* parameter to reference the OPOS **BinaryConversion** property, see page 583.
- Added a Sequence Diagram to the POS Keyboard device chapter, see page 593.
- Added a Sequence Diagram to the POS Power device chapter, see page 607.
- Updated/clarified the text in the various diagrams in the POS Power Chapter.
- Added clarification of the pixel handling capability of the POS Printer, see “Capabilities” on page 627.
- Updated the Class Diagram of the POS Printer device chapter, see page 629.
- Added a new ESC sequence to the POS Printer device chapter providing for more reliable handling of pass through data, see page 638 and page 639.
- Updated the POS Printer device to support CodePage mapping by adding the **CapMapCharacterSet** (see page 653) and **MapCharacterSet** (see page 673) properties.
- Updated the POS Printer device to add support for printing Barcodes and Bitmaps to **rotatePrint** by adding the **RecBitmapRotationList** (see page 674) and **SlpBitmapRotationList** (see page 682) properties, and updating the **SlpBarcodeRotationList** (see page 681) property.
- Added additional meaning for the E_ILLEGAL error in the **printBarcode** method of the POS Printer, see page 704.
- Clarified the format of the file referenced by the *fileName* parameter of the **printBitmap** method of the POS Printer for the OPOS environment, and clarified the interaction between mixed text and bitmap printing, see page 705.

- Updated the following POS Printer methods/parameter to reference the OPOS **BinaryConversion** property:

• printBarCode	<i>data</i>	see page 701.
• printImmediate	<i>data</i>	see page 708.
• printNormal	<i>data</i>	see page 710.
• printTwoNormal	<i>data1/data2</i>	see page 712.
• setLogo	<i>data</i>	see page 718.
• validateData	<i>data</i>	see page 721.
- Expanded the allowable values of the *bitmapNumber* parameter of the **setBitmap** method of the POS Printer, see page 717.
- Clarified the format of the file referenced by the *fileName* parameter of the **setBitmap** method of the POS Printer for the OPOS environment, and clarified the interaction between mixed text and bitmap printing, see page 717.
- Updated the Remote Order Display device to support CodePage mapping by adding the **CapMapCharacterSet** (see page 741) and **MapCharacterSet** (see page 747) properties.
- Updated the Remote Order Display **displayData** method *data* parameter to reference the OPOS **BinaryConversion** property, see page 758.
- Added a Sequence Diagram to the Scale device chapter, see page 778.
- Updated the Scale **displayText** method *data* parameter to reference the OPOS **BinaryConversion** property, see page 785.
- Added a Sequence Diagram to the Scanner device chapter, see page 795.
- Updated the Scanner **ScanData** (see page 798) and **ScanDataLabel** (see page 799) properties to reference the OPOS **BinaryConversion** property.
- Added a Sequence Diagram to the Signature Capture device chapter, see page 810.
- Updated the Signature Capture **PointArray** (see page 815) and **RawData** (see page 816) properties to reference the OPOS **BinaryConversion** property.
- Added a Sequence Diagram to the Tone Indicator device chapter, see page 857.
- Made the OPOS Windows operating Systems supported a more general statement, and added the exclusion of Windows 3.x, removed reference to the deliverable of the CPG, see Appendix A, page A-1.
- Added an Event Registration Sequence Diagram, see Appendix A, page A-13.
- Added a language specific Common Properties Summary Table to the OPOS Appendix, see Appendix A, page A-24.
- Added a language specific Programmatic Names Table to the OPOS Appendix, see Appendix A, page A-26.
- Added table to the **BinaryConversion** property description to define the affected devices and properties/methods, see Appendix A, page A-28.
- Added **CapStatusMultiDrawerDetect** to the two tables describing the Cash Drawer Properties Operations, starting on Appendix A, page A-59.
- Added an asterisk to identify **OpenDrawer** as required for basic operations to the two tables describing the Cash Drawer Properties Operations, starting on Appendix A, page A-59.

- Added Check Scanner and Motion Sensor to the Device Class Keys list, see Appendix A, page A-68.
- Added Check Scanner and Motion Sensor to the Header Files list, see Appendix A, page A-72.
- Added Code Page technical information regarding the Mapping of **CharacterSet**, see Appendix A, page A-75.
- Added the original OPOS Application Programmers Guide Change History for Revisions 1.01 through 1.6, see Appendix A, page A-77.
- Added the OPOS Control Programmers Guide as Section 8, see Appendix A, page A-89.
- Added an Event Registration Sequence Diagram, see Appendix B, page B-18.
- Updated the JavaPOS Package Structure descriptions, also added CheckScanner and MotionSensor devices, see Appendix B, page B-37
- Added a language specific Common Properties Summary Table to the JavaPOS Appendix, see Appendix B, page B-58.
- Added a language specific Class Names Table to the JavaPOS Appendix, see Appendix B, page B-59.
- Added clarification of the initial value of the **PowerNotify** property after the **open** method call, see Appendix B, page B-67.
- Added **CapStatusMultiDrawerDetect** to the table describing the Cash Drawer Properties Operations, see Appendix B, page B-84.
- Added an asterisk to identify **openDrawer** as required for basic operations to the tables describing the Cash Drawer Properties Operations, see Appendix B, page B-84.
- Added Code Page technical information regarding the Mapping of **CharacterSet**, see Appendix B, page B-91.
- Added the original JavaPOS Programming Guide Change History for Revisions 1.3 through 1.6, see Appendix B, page B-92.
- Added reference detailing 2nd USB PlusPower connector, reworded the description of the PlusPower connectors, and added information on the IBM patents, see Appendix E, page E-1.
- Made minor typographical and formatting changes throughout the document as necessary.

Release Version 1.8

Version 1.8 of this specification, released on June 30, 2003, includes a new chapter describing the Smart Card Reader Writer device, additions for the support of Device Statistics that affect every device/chapter, and contains several updates to the existing chapters that provide enhancements, clarifications, and corrections to Version 1.7. These changes are detailed below, with links to the corresponding sections, pages, or chapters as appropriate. However, any minor typographical changes are not listed below.

- Updated the Version and issue date on the front [page](#).
- Added new company names to the Member list, see [page iii](#).
- Updated the Table of Contents to reflect additional sections. See “[TABLE OF CONTENTS](#)” on [page i](#).
- Added the Device Statistics information to the Introduction and Architecture Chapter see [page 26](#), Common PME Chapter see [page 33](#), [page 36](#), [page 39](#), and [page 54](#), all the device Chapters in the Summary Tables, and the OPOS and JavaPOS Appendices also in the Summary Tables, and Properties and Methods Sections.
- Updated several Sequence Diagrams in order to more closely depict the sequence of the Service processing of event firing and the decrement of **DataCount**. Updated diagrams are in the MICR, MSR, POSKeyboard, Scanner, and SignatureCapture chapters.
- Reworded the handling of Workstation or POS terminal power loss support under the Device Power Reporting Model, see [page 22](#), [Appendix A](#), [page A-17](#), and [Appendix B](#), [page B-24](#).
- Corrected minor typographical error in and reformatted the layout of the CashChanger State Diagram, see [page 101](#).
- Corrected the Summary section definition of parameters of the Cash Drawer **openDrawer** and **waitForDrawerClose** methods, see [page 122](#), and Class Diagram, see [page 124](#).
- Corrected the ErrorResponse type of the CAT ErrorEvent to read-write, see [page 171](#).
- Added various enhancements to the Model discussion for the Fiscal Printer, starting on [page 240](#).
- Updated the **Fiscal Receipt** and **Fiscal Receipt Ending** descriptions of the Fiscal Printer to allow use of the **printRecMessage** method in these states, see [page 242](#).
- Updated the **Message Lines** description of the Fiscal Printer Receipt Layouts, see [page 247](#).
- Updated the **CapAdditionalLines** property of the Fiscal Printer, see [page 254](#).
- Expanded the description of PTR_SUE_SLP_EMPTY status of the Fiscal Printer StatusUpdateEvent, see [page 366](#).
- Added support for multiple covers in the Fiscal Printer StatusUpdateEvent, see [page 367](#).
- Clarified the wording of the **claimFile** method in the HardTotals device, see [page 382](#).

- Added DISP_CCT_BLINK to the LineDisplay **CapCursorType** capability, see [page 415](#).
- Added DISP_CT_BLINK to the LineDisplay **CursorType** property, see [page 421](#).
- Corrected the wording in the PINPad Features not Supported section, last bullet, to remove the word “**not**”, see [page 516](#).
- Corrected the type of the PINPad device’s **Amount** property from *int32* to *currency* in both the Summary and Properties sections, see [page 510](#) and [page 519](#).
- Corrected the ErrorResponse type of the PINPad ErrorEvent to read-write, see [page 536](#).
- Clarified the pixel-level addressing for the POSPrinter, see [page 627](#).
- Added various enhancements to the Model discussion for the POSPrinter, starting on [page 630](#).
- Added clarification in POSPrinter describing cartridge statuses, see [page 632](#).
- Added discussion in POSPrinter describing actions of partial line printing, see [page 635](#).
- Corrected the ESC sequence for Feed and Paper Cut in the POSPrinter device, see [page 639](#).
- Updated the four POSPrinter Low Level state diagrams, starting on [page 641](#).
- Added clarification to the handling and printing of the PTR_BCS_Code128 barcode format supported by the POSPrinter device, **printBarCode** method, see [page 701](#).
- Added additional RSS barcode formats supported by the POSPrinter device **printBarCode** method, see [page 702](#).
- Added clarification of status of **RotateSpecial** and usage of PTR_RP_BARCODE under **rotatePrint** in POSPrinter, see [page 715](#).
- Expanded the description of PTR_SUE_SLP_EMPTY status of the POSPrinter StatusUpdateEvent, see [page 726](#).
- Added support for multiple covers in the POSPrinter StatusUpdateEvent, see [page 727](#).
- Clarified the check digit handling for the **ScanDataLabel** property supported by the Scanner device, see [page 799](#).
- Added additional RSS **ScanDataType** formats supported by the Scanner device, see [page 800](#).
- Added the chapter describing the Smart Card Reader Writer device. See “CHAPTER 24 SMART CARD READER / WRITER” on [page 823](#). The chapters following have been renumbered accordingly.
- Moved the Tone Indicator chapter from 24 to 25 to make room for the Smart Card Reader Writer chapter that is added in this release.
- Made the wording consistent in the OPOS Appendix Methods (except **Open**), Return section.
- Added Smart Card Reader Writer to the OPOS Programmatic Names list, see [Appendix A, page A-26](#).
- Added Smart Card Reader Writer to the Device Class Keys list, see [Appendix A, page A-68](#).
- Added Smart Card Reader Writer to the Header Files list and corrected MotionSensor file name to match released file name, see [Appendix A, page A-72](#).

- Added Smart Card Reader Writer to the Internal Header Files list and corrected MotionSensor file name to match released file name, see [Appendix A, page A-122](#).
- Updated the JavaPOS Package Structure descriptions, also added the Smart Card Reader Writer device, see [Appendix B, page B-37](#).
- Corrected the package names for PointCardRWService15 through PointCardRWService17 and POSPowerService15 through POSPowerService17, see [Appendix B, page B-41](#).
- Added Smart Card Reader Writer to the JavaPOS Class Names, see [Appendix B, page B-60](#).

A P P E N D I X D

Additional Software References

This appendix contains a list of additional material that may prove helpful for the understanding of the UnifiedPOS software environment.

UML References

The following 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

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

Overview

USB, or the Universal Serial Bus, is a communications attachment standard that includes power in the cable connection to the peripheral device. One of the limitations of USB is the amount of +5 volt current available to supply attached peripherals. Normally, 500 milliamp is available at each host port and each powered external hub port. This amount of current is sufficient for most PC type peripherals like mice and keyboards. When the power requirements exceed the 500 milliamp limitation, external peripherals require the use of an external power supply (brick) to supply the necessary power requirements. This limitation takes away from the true “plug-n-play” idea conceived for USB peripherals.

The PlusPower USB connector provides a single cable connection that supplies both the standard USB communication signals and two additional wire pairs for extra power.

Host Side Connector

The host connector incorporates an “A” type socket that allows compatibility of standard USB peripherals. The connector itself is unique in that it provides the additional benefit of a locking mechanism for the cable connector. The host connector’s four power pins (two ground and two voltage) are keyed to a specific voltage available at that port.



The following voltage keying options are available:

- +5 volts DC at a maximum rating of 6 amps per connector
- +12 volts DC at a maximum rating of 6 amps per connector
- +24 volts DC at a maximum rating of 6 amps per connector

Cable

The cable end is also keyed to match the voltage type and is color coded to simplify voltage identification.



- +5 volts (ivory)
- +12 volts (teal)
- +24 volts (red)

Peripheral Side Connection

The peripheral side connection is loosely defined and generally left to the specific user's physical space requirements. The Series B connector as supplied by FCI/Berg is the recommended connector but not mandatory.

Web Location References - USB connector EIA approval

- Approved March 2000 as EIA standard.
- Defines 12 and 24 volt key connectors.
- EIA 700BAAD number assigned.

Official On-line Documentation for the USB PlusPower connector is available at:

<http://www.eia.org>

http://www.tiaonline.org/standards/search_n_order.cfm

Reading Material References

- 1) EIA-700BAAD, Detail Specification for Shielded Rectangular Connector(s) For Universal Serial Bus PlusPower Connector(s) Series "A", EIA Engineering Publications Office, 2500 Wilson Boulevard, Arlington, Virginia, 22201.
- 2) EIA-700BAAE, Detail Specification for Shielded Rectangular Connector(s) For Universal Serial Bus PlusPower Connector(s) Series "B", EIA Engineering Publications Office, 2500 Wilson Boulevard, Arlington, Virginia, 22201.

ARTS Standard Endorsement

ARTS has adopted the Powered USB connectors (as defined in EIA Standard EIA-700BAAD and EIA-700BAAE) as a retail standard for attachment of point-of-sale I/O devices. This is in keeping with the following ARTS objectives:

- Provide the retail community with a widely available connection standard that increases options and function while reducing cost
- Protect the retail community from legal actions or restrictions that might hinder operations, limit future options, or increase costs

In response to this endorsement of technology which includes an IBM patent, IBM is pleased to offer a royalty free license for Point-Of-Sale usage of the powered USB connector as described in the following statement:

“IBM will make available to retail point-of-sale vendors, a non-exclusive fully paid-up license under U.S. Patent No.: 6,086,430 (and any corresponding patents of other countries) to use Powered USB connectors (as defined in EIA Standard EIA-700BAAD and EIA-700BAAE) in Retail point-of-sale terminals, upon the signing of a license agreement and payment of a nominal fee.”

The fee referenced is \$5,000 per ARTS member as the one time charge for the patent.

For the patent license please contact:

Director of Licensing
International Business Machines Corporation
North Castle Drive
Armonk, New York 10504-1785

The agreement provides a license to products which are considered a Point-of-Sale Device or a peripheral device designed primarily for attaching to a Point-of-Sale Device; and, which contain connectors which conform to and operate in compliance with specifications for a Supported Standard. A Point-of-Sale Device means a device designed primarily for use in retail stores for recording sales data and handling on-site customer transactions at the time a sale is made. A Supported Standard is defined as the Detail Specification for Shielded Rectangular Connectors for Universal Serial Bus Plus Power Connectors Series “B” (ANSI/EIA-700BAAE-00) (Published: May 9, 2000) and/or Detail

Specification for Shielded Rectangular Connectors for Universal Serial Bus Plus Power Connectors Series "A" (ANSI/EIA-700BAAD-00) (Published: May 10, 2000). This is a limited field of use licensing arrangement, available for a one time fee of \$5000 from IBM, for applications determined by IBM to be compliant with the license definitions referenced above. All other uses of these patents, in support of specifications or standards, are available from IBM under non-exclusive, non-discriminatory, reasonable terms and conditions, in accordance with IBM's normal licensing policies. The license is available to Point-of-Sale manufacturers, value added resellers, and systems integrators.