



WS-POS Technical Specification

Version 1.1

June 15, 2011 Release

Chairman UnifiedPOS Committee:

H Paul Gay	Epson America
------------	---------------

UnifiedPOS Committee Members:

Gerald Armentrout	IBM
Kunio Fukuchi	Fujitsu Frontech Limited
Tadashi Furuhata	SEIKO EPSON Corporation
Denis Kuniss	Wincor-Nixdorf
Vic Miles	Microsoft
Curtiss Monroe	NCR Corporation
Jürgen Moser	Bizerba
Lawrence Owen	Star Micronics Co., Ltd.
Brian Spohn	NCR Corporation
Michael Webb	Data Logic, Inc.

Contributors:

Richard Halter	NRF-ARTS
----------------	----------

WS-POS 1.1 Technical Specification

Work Team For WS-POS Version 1.1

Chairmen OPOS-J Work Team:

Toru Yanagisawa	NEC Infrontia Corporation.
Masanori Sambe	Toshiba TEC Corporation
Takao Tamura	Sorimachi Giken Co., Ltd.

Core OPOS-J Members:

Tadashi Furuhata	SEIKO EPSON Corporation
Hideo Nakamura	SEIKO EPSON Corporation
Kunio Fukuchi	Fujitsu Frontech Limited
Toyohiro Yasumoto	Vinculum Japan Corporation
Kiyotaka Abe	Sorimachi Giken Co., Ltd.
Kenichi Nagai	Star Micronics Co., Ltd.
Yuji Mori	Star Micronics Co., Ltd.
Akio Tajima	NCR Japan, Ltd.
Takahiro Akutsu	Hitachi Information & Communication Engineering, Ltd.

Contributors OPOS-J:

Soichi Fujii	Microsoft KK
Hiroshi Ota	Microsoft KK
Mitsuo Nagata	Vinculum Japan Corporation

Copyright © National Retail Federation 2011. All rights reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be

WS-POS 1.1 Technical Specification

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 the 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.

TABLE OF CONTENTS

1. ABSTRACT.....	8
1.1 OVERVIEW	8
1.2 CONFORMANCE REQUIREMENTS	9
1.3 ARTS WS-POS STANDARDS STACK	10
1.4 OUT OF SCOPE.....	11
2. WS-POS COMPONENTS.....	12
2.1 EXPLANATION OF WS-POS RELATED TERMINOLOGY	12
2.2 MESSAGING BASE	13
2.3 SERVICE DESCRIPTION AND DISCOVERY	14
2.4 STATUS, STATE MODEL AND EXCEPTIONS	18
2.5 SHARED DEVICE MODEL.....	20
2.6 EVENT MESSAGES.....	21
2.7 INPUT MODEL	22
2.8 OUTPUT MODEL.....	25
2.9 DEVICE POWER REPORTING MODEL	26
2.10 ARTS XMLPOS COMMAND SET	27
2.11 ARTS XMLPOS EVENT SET	28
2.12 ARTS XMLPOS SCHEMA	29
2.13 WS-POS WSDL	30
2.14 DISCOVERY	34
2.15 EVENTS	37
2.16 SECURITY	41
2.17 XML PAYLOAD	43
3. GENERAL FLOW.....	44
3.1 GENERAL WS-POS FLOWS	44
3.2 SIMPLE USE CASE	46
3.3 USE CASE CATALOG	50
3.4 SCOPE	50
3.5 SUB SCOPE: LINKAGE BETWEEN IN-STORE KIOSK AND POS (DEVICES)	54
3.6 SUB SCOPE: LINKAGE BETWEEN SALES ASSISTANCE TERMINALS AND POS DEVICES	59
3.7 SUB SCOPE: BATCH PAYMENT IN COMMERCIAL COMPLEX.....	66
3.8 SUB SCOPE: MONITORING IN-STORE KIOSK EQUIPMENT AND COOPERATION WITH BACK-OFFICE ON OCCURRENCE OF PROBLEMS.	70

WS-POS 1.1 Technical Specification

3.9 SUB SCOPE: POS SYSTEM IN CONSIDERATION OF COOPERATION BETWEEN VARIOUS INDUSTRIES	77
3.10 SUB SCOPE: LINKAGE BETWEEN DISPLAY SHELF AND REAR SYSTEM	81
3.11 SUB SCOPE: COOPERATION BETWEEN ELECTRONIC SHELF LABEL AND SHELVING ALLOCATION INFORMATION	89
3.12 SUB SCOPE: SELF-SERVICE REFUELING	94
4. DOCUMENT HISTORY.....	102
5. GLOSSARY.....	103
6. REFERENCED DOCUMENTS AND SOFTWARE SUPPORT FILES	104
6.1 REFERENCED DOCUMENTS.....	104
6.2 SOFTWARE SUPPORT FILES	105
7. WS-POS CLASS DIAGRAMS	106
BELT.....	107
BELT EVENTS	111
BILL ACCEPTOR.....	112
BILL ACCEPTOR EVENTS	116
BILL DISPENSER	117
BILL DISPENSER EVENTS	121
BIOMETRICS	122
BIOMETRICS EVENTS	127
BUMP BAR.....	129
BUMP BAR EVENTS	133
CASH CHANGER.....	135
CASH CHANGER.....	139
CASH CHANGER EVENTS	141
CASH DRAWER	142
CASH DRAWER EVENTS.....	145
CAT (CREDIT AUTHORIZATION TERMINAL).....	146
CAT (CREDIT AUTHORIZATION TERMINAL) EVENTS	152
CHECK SCANNER	154
CHECK SCANNER EVENTS	161
COIN ACCEPTOR	163
COIN ACCEPTOR EVENTS.....	169
COIN DISPENSER.....	170
COIN DISPENSER EVENTS	174

WS-POS 1.1 Technical Specification

ELECTRONIC JOURNAL	175
ELECTRONIC JOURNAL EVENTS	181
ELECTRONIC VALUE R/W (ELECTRONIC VALUE READER AND WRITER).....	183
ELECTRONIC VALUE R/W (ELECTRONIC VALUE READER AND WRITER) EVENTS	189
FISCAL PRINTER	191
FISCAL PRINTER EVENTS	206
GATE	208
GATE EVENTS	211
HARD TOTALS	212
HARD TOTALS EVENTS.....	216
IMAGE SCANNER	217
IMAGE SCANNER EVENTS	221
ITEM DISPENSER	223
ITEM DISPENSER EVENTS	227
KEYLOCK	228
KEYLOCK EVENTS	232
LIGHTS	233
LIGHTS EVENTS	237
LINE DISPLAY	238
LINE DISPLAY EVENTS	245
MAGNETIC INK CHARACTER RECOGNITION (MICR)	246
MAGNETIC INK CHARACTER RECOGNITION (MICR) EVENTS	251
MOTION SENSOR	253
MOTION SENSOR EVENTS	257
MAGNETIC STRIPE READER (MSR)	258
MAGNETIC STRIPE READER (MSR) EVENTS.....	266
PERSONAL IDENTIFICATION NUMBER PAD (PIN PAD)	268
PERSONAL IDENTIFICATION NUMBER PAD (PIN PAD) EVENTS	274
POINT CARD READER/WRITER (POINT CARD R/W).....	276
POINT CARD READER/WRITER (POINT CARD R/W) EVENTS	284
POS KEYBOARD	286
POS KEYBOARD EVENTS	290
POS POWER.....	292
POS POWER EVENTS	297
POS PRINTER	299
POS PRINTER EVENTS	313

REMOTE ORDER DISPLAY.....	315
REMOTE ORDER DISPLAY EVENTS	323
RADIO FREQUENCY IDENTIFICATION SCANNER (RFID SCANNER)	325
RADIO FREQUENCY IDENTIFICATION SCANNER (RFID SCANNER) EVENTS.....	330
SCALE.....	332
SCALE EVENTS	336
SCANNER (BAR CODE)	338
SCANNER (BAR CODE) EVENTS	342
SIGNATURE CAPTURE	344
SIGNATURE CAPTURE EVENTS.....	348
SMART CARD READER/WRITER (SMART CARD R/W)	350
SMART CARD READER/WRITER (SMART CARD R/W) EVENTS	355
TONE INDICATOR.....	357
TONE INDICATOR EVENTS	362
APPENDIX A APPLICATION DEVELOPMENT SUPPORT	364

TABLE OF FIGURES

FIGURE 1: WEB SERVICES ARCHITECTURE.....	9
FIGURE 2: ARTS WS-POS STACK	10
FIGURE 3: ARTS SCHEMA - WSDL - UDDI RELATIONSHIP.....	14
FIGURE 4: WSDL OVERVIEW	31
FIGURE 5: INTEROPERABILITY AND PLATFORM EQUIVALENCE TO THE CODE CONVERSION.....	32
FIGURE 6: UNIFIEDPOS AND WSDL MAPPING.....	33
FIGURE 7: UDDI OVERVIEW	35
FIGURE 8: REGISTERING A WSDL WITH A UDDI REGISTRY.....	36
FIGURE 9: EVENT REGISTRATION AND DEREGISTRATION	38
FIGURE 10: AUTOMATIC EVENT REGISTRATION/DEREGISTRATION DURING OPEN	40

1. ABSTRACT

1.1 Overview

The W3C Glossary, <http://www.w3.org/TR/ws-gloss/>, defines a Web Service as “a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-understandable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.”

Web Services for Point of Service (WS-POS) is a technical document intended to provide retail devices, terminals and servers with the capabilities necessary to interoperate in a detached, dynamic network as well as more typical retail LANs by leveraging these W3C specifications.

WS-POS Version 1.1 represents an update of the original release of the standard with emphasis on refined WSDL support for all UnifiedPOS Version 1.13 peripheral devices as well as enhancements and bug fixes. This amended material was created almost exclusively as a result of implementation experience and subsequent refinement submissions from the OpenPOS Technology Council of Japan.

A significant part of this document is a profile of the minimal web-service specifications necessary to support remote device interoperability. This profile serves to constrain, articulate, and enable the usage of the WS specifications in order to facilitate interoperability and ensure appropriateness of the chosen web-services specifications for a Web Services implementation at the retail store. The WS-POS specification also utilizes UnifiedPOS XMLPOS and other significant details intended to permit an easy and interoperable implementation of a Web Service for Point of Service solution.

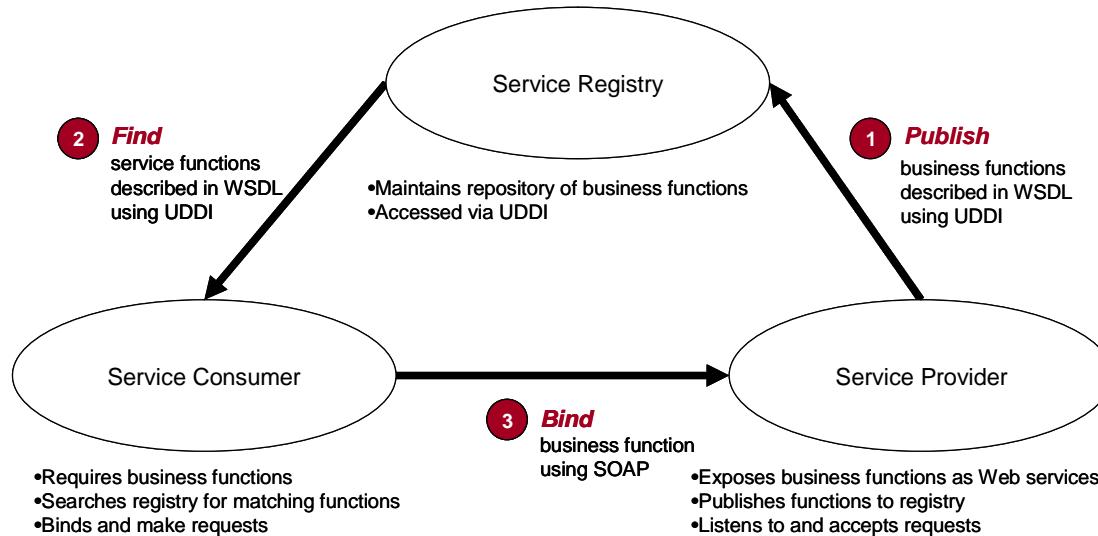


Figure 1: Web Services Architecture

Figure 1 illustrates a typical connectivity model for Web Services in general, and more importantly, the one that will be used for WS-POS. First, a Service Provider registers with a Service Registry (UDDI) identifying the services it provides. Second, the Service Consumer finds the locations of the Service Providers who supply the services it needs to perform its task. Finally, it binds directly to the identified Service Providers to exchange the necessary messages to perform that service.

1.2 Conformance Requirements

In order to ensure interoperability as defined by this specification, an implementation must meet all the requirements set forth in this specification. These requirements are defined using the following format:

Id	Name	Description
XY001	Requirement Name	This is a requirement description

1.3 ARTS WS-POS Standards Stack

The following diagram shows the WS-POS stack components necessary to be compliant with this standard.

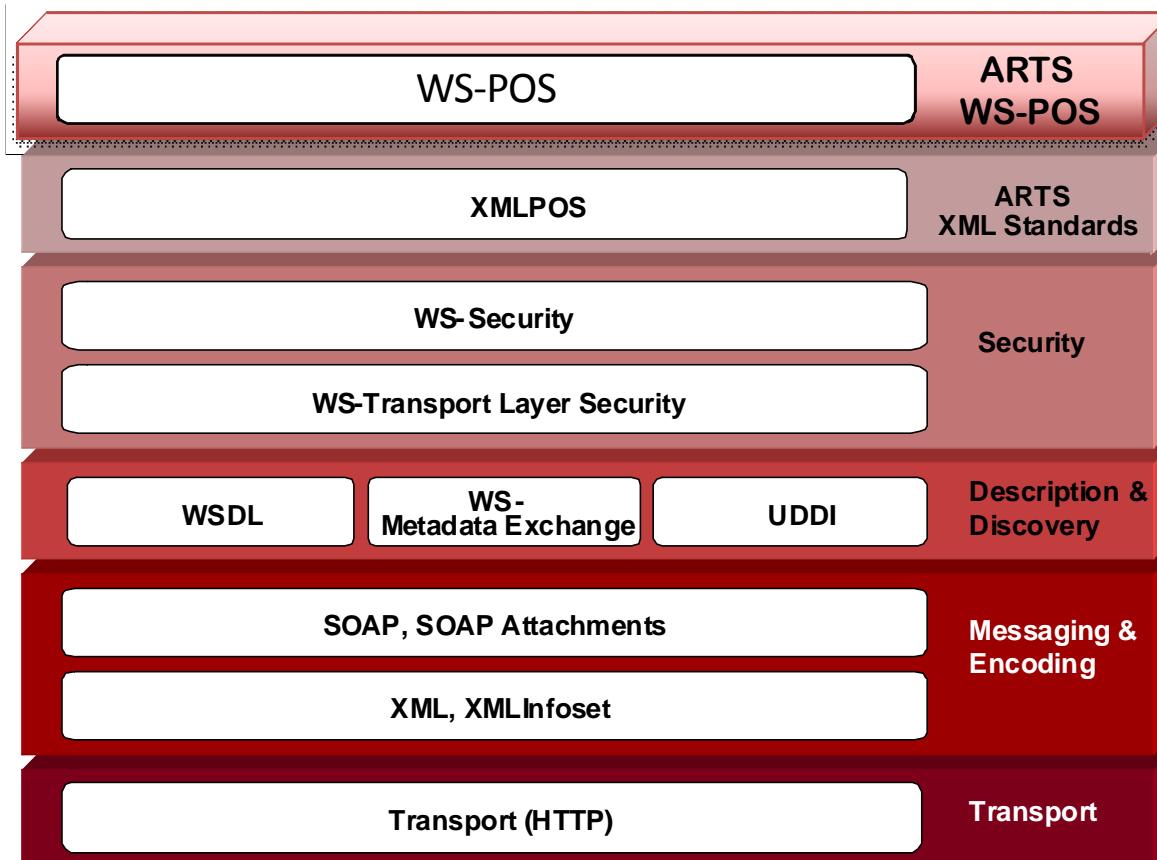


Figure 2: ARTS WS-POS Stack

1.3.1 Web Service Components

The web services definition of WS-POS addresses three key areas:

- a) The Messaging base: a description of the core that all messaging in WS-POS is based upon.
- b) Service discovery: How resources and services are located.
- c) Service description: How services and messages of interest will describe themselves.

1.3.2 Additional Components

- a) Security: provides a description of the mechanisms that must be in place to ensure an appropriate level of security.
- b) XML Payload: The payload of a message.
- c) Service Definitions: In order to enable clients to easily find available devices using Universal Data Discovery and Integration (UDDI), it is necessary to define the names and definitions of device services.

1.4 Out of Scope

Management, dynamic service discovery, and events at the web service level have not been addressed in this version of the specification.

This version limits itself to data flow within the “retail store” environment. Future versions may expand in scope to include data flow to and from the enterprise. This has the implication that all security outside of interactions between UDDI, Client, and Device require implementation practices considered under the guidance of good systems design practices.

This specification does not specify:

- Methods for selecting a device service to use from a list of possible devices services. This is an area an application needs to manage.
- Methods to preload devices with security credentials or validate security credentials.
- How to verify that devices or clients are valid.
- Address Payment Card Industry (PCI) data requirements. It is recognized that the information contained within the messages will need to follow the PCI standard. Those requirements are defined by the PCI committee.
- Methods for handling unintended disconnect scenarios between a WS-POS Service Provider and a Service Consumer, especially after registration of EventDeviceListeners (see section 2.15 “Events”), are out of scope. See section 6 for additional helpful references.

2. WS-POS COMPONENTS

This section provides details concerning each component of the WS-POS stack.

2.1 Explanation of WS-POS Related Terminology

2.1.1 XML POS

XML POS is a definition in XML of the operations, states, and attributes that should be supported by POS devices standardized by ARTS.

2.1.2 Web Service

A Web Service provides a standard method of interoperability for different software to operate on various platforms (W3C). Many of the implementations are defined in the WSDL definition language and use a protocol to exchange information in XML format called SOAP while using HTTP for transport. In some cases Web Service refers to the technology itself while in other cases it refers to services implemented by the technology.

2.1.3 Web Service based on the WS-POS specification

This specification describes a Web Service based on the WS-POS specification as a WS-POS service.

2.1.4 Provider and consumer of a WS-POS service

This specification describes the WS-POS service provider who is providing the WS-POS service and WS-POS service consumer who is using the WS-POS service.

2.1.5 Methods in WS-POS

Methods in WS-POS are expressed as Web Service operations. This specification uses WS-POS methods instead of the term of Web Service operations.

2.1.6 Properties in WS-POS

Properties in WS-POS are realized using Getter/Setter methods. For example, the **DeviceEnabled** property defined by UPOS is expressed by the **GetDeviceEnabled** method as a Getter method while the **SetDeviceEnabled** method is expressed as a Setter method. In read-only properties, there are no Setter methods. This specification typically does not describe Getter method/Setter methods hereafter. Please note that Getter/Setter methods are used to acquire/set properties.

2.1.7 Events in WS-POS

Events in WS-POS notify WS-POS service consumers about device events defined by Unified POS from WS-POS service providers.

WS-POS calls the client from the Web Service in order to receive device events defined in Unified POS by the application. Applications that receive events must notify the services about the end point for reception.

2.2 Messaging Base

2.2.1 Description

The messaging base is the set of technologies utilized as a core basis for all the other areas. It provides a starting point for description of all the messages that are employed.

2.2.2 WS-I Basic Profile

A WS-POS service is described by WSDL 1.1, based upon WS-I Basic Profile version 1.2. This consists of using SOAP 1.1 over HTTP, with WS-Addressing Core 1.0, and Web Service Description Language (WSDL) Version 1.1. The WSDL 1.1 description includes the service offered (type/interface), method of use (bind), and location (endpoint). These are the mainstream defaults for web service usage, and are typically seen as the “ground rules” by which the Web Services “world” operates.

Id	Name	Description
MB001	Basic Profile	An implementation MUST be conformant to WS-I Basic Profile version 1.2

2.3 Service Description and Discovery

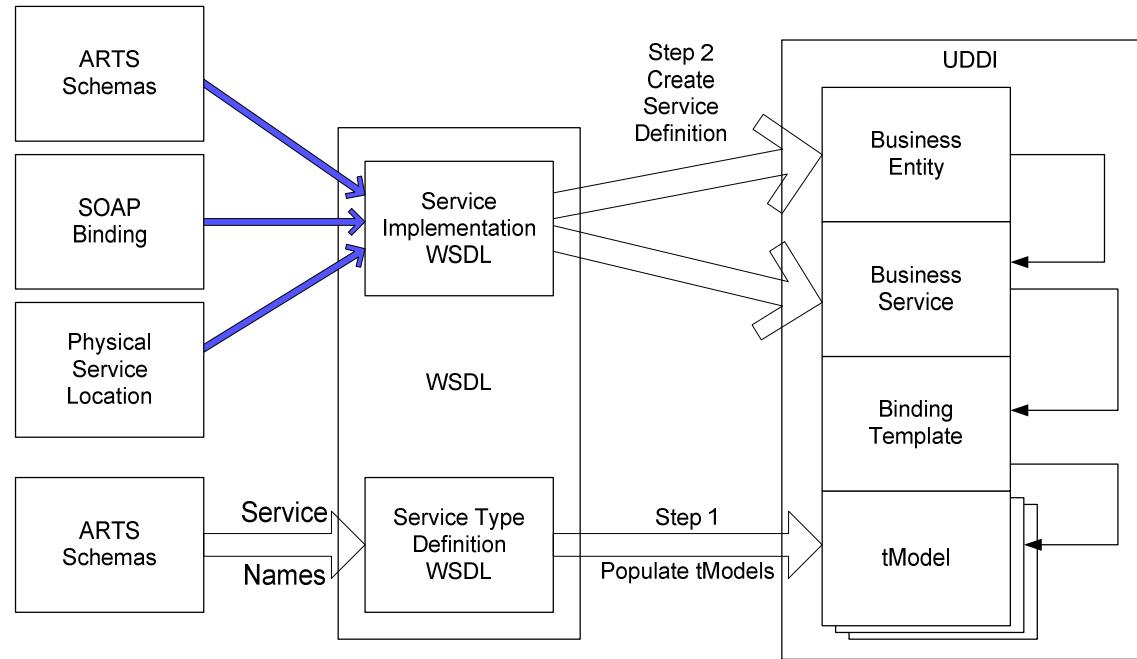


Figure 3: ARTS Schema - WSDL - UDDI Relationship

Populating a UDDI registry is a two step operation. First, the tModel (Technical Model) is populated from a WSDL with the appropriate Service information, usually provided by a Service Provider. Then, the Business Service is populated with links to the group of tModels necessary to perform a specific Business Service.

From the registry a Service Consumer may then look up the Business Services it needs to perform its task. Once an appropriate Business Service is determined, the Service Consumer may query the UDDI registry for contact information for each individual Web Service of that type.

UDDI 3.0.2 can be used to discover WS-POS services. WS-POS service providers register WS-POS services in the UDDI service registry.

The relationships between the WSDL description and the information registered in UDDI are the noted as follows:

WSDL	UDDI
types	tModel
message	
binding	
Service	businessService
Port	bindingTemplate

In addition, the ARTS Device Type (Scanner, POSPrinter, etc.) is set in businessEntity of UDDI.

2.3.1 Service Description

Description

This area defines the techniques that will be used to formally describe the service being offered in a standard way such that all parties can have an agreed-upon view as to the particulars of the messages exchanged for this purpose. Along with a good description base comes the ability to have strong tools to assist in the development and verification of conforming products.

Metadata

WSDL 1.1, as explained in WS-I Basic Profile V1.2, is employed as the core service description facility. WS-MetaDataExchange is required to retrieve the metadata information relevant to the respective Web services utilized.

2.3.2 WS-POS Service Use (Methods and Properties)

In order to enable a WS-POS service consumer to use the methods and the properties of a WS-POS service, it refers to end point URL of the WS-POS service provider in accordance with WSDL and calls the necessary WS-POS methods.

In modern programming, for example, in the WCF of .NET and JAX-WS of Java, the proxy code of the WS-POS service is generated from WSDL and the schema, and the WS-POS service consumer uses the WS-POS service via the proxy code.

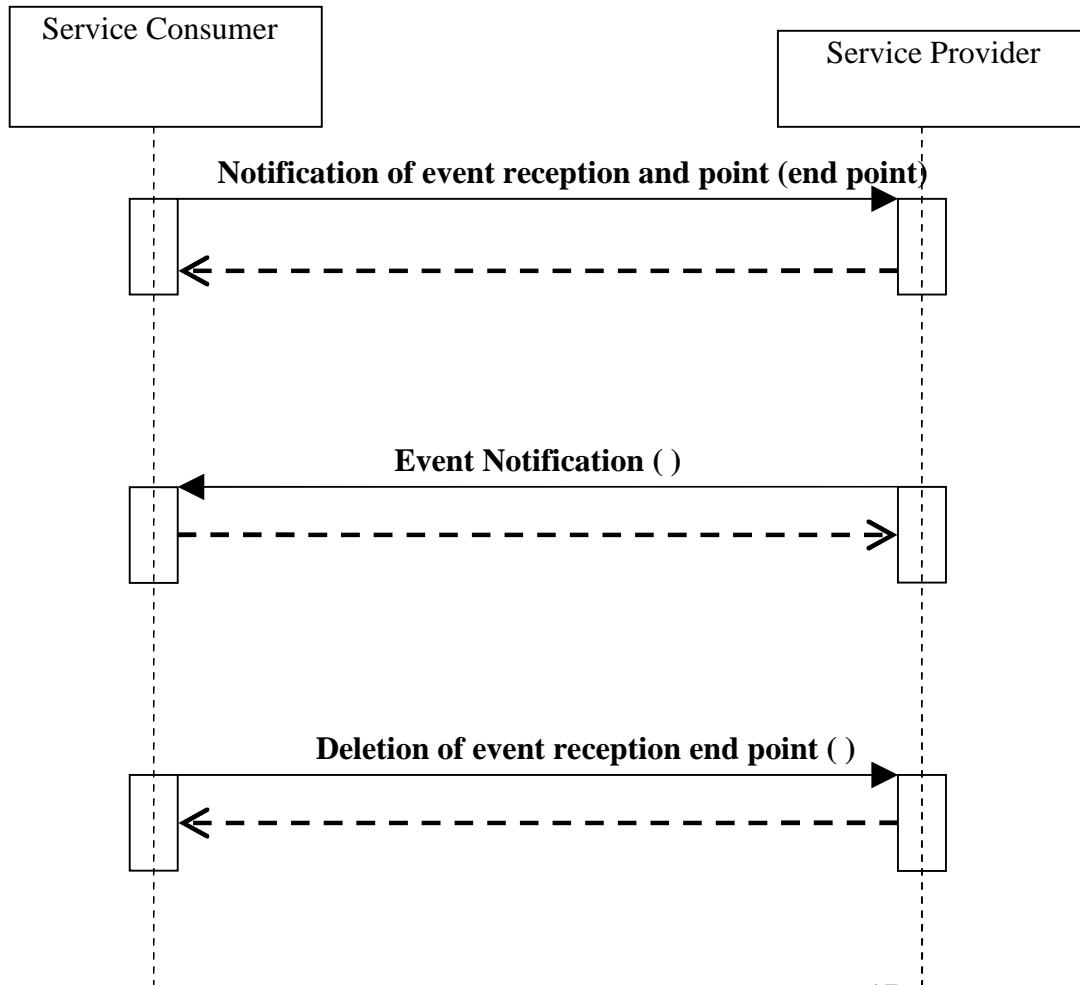
2.3.3 WS-POS Service Use (Events)

WS-POS service providers may notify WS-POS service consumers of device events defined by Unified POS. WS-POS service consumers who would like to receive the events for this notification must notify the WS-POS service provider of the end point for event reception.

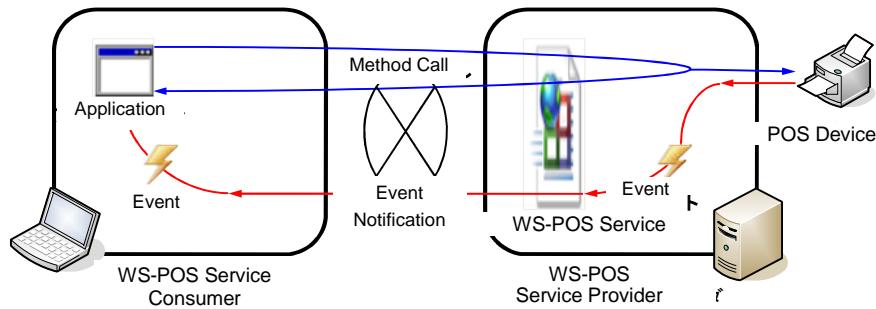
The WS-POS service provider notifies the Unified POS device event to the end point for event reception notified by the WS-POS service consumer.

The WS-POS service consumer notifies the WS-POS service provider to delete the event reception end point when event reception is no longer necessary.

A sequence chart of the procedure is shown below.



In addition, an outline of the method call and event notification is shown below.



The end point notification by the WS-POS service consumer for this event reception and the reception by the WS-POS provider receive support from the programming environment and are realized by the proxy code and the configuration file.

2.4 Status, State Model and Exceptions

As shown below, there are some commonly used enumerations, events, and properties in which the status, error code, and state model are set.

2.4.1 StatusUpdateEvent

StatusUpdateEvent is an event generated when the state unique to the specific class and status variable are changed.

2.4.2 ControlState

ControlState is an enumeration to store the current state. Possible values are below.

Closed
Idle
Busy
Error

2.4.3 Exceptions

There is the possibility that all WS-POS service method calls throw **UposException** when they fail.

UposException is defined in each device under the namespace <http://www.nrf-arts.org/UnifiedPOS/> by the XMLPOS schema of ARTS.

2.4.4 Public Properties

Name	Description
ErrorCode	ErrorCode is an error code which represents the cause of the error exception and is defined by an enumeration.
ErrorCodeExtended	ErrorCodeExtended is an extension error code which represents the cause of the error exception. Values unique to the service can be stored in this code.

2.5 Shared Device Model

The shared device model of WS-POS supports devices that can be partially or completely shared by multiple applications (WS-POS service consumer) as well as devices that can only be exclusively used by one WS-POS service consumer simultaneously. All WS-POS service providers are open to one or more applications. However, some behaviors that can be executed in a WS-POS service provider limit applications that acquire access rights to a device to only one application.

2.5.1 Exclusive Use Device

The most common device types are “Exclusive Use Devices” and one such example is the POS printer. This device can be used simultaneously by one application only (WS-POS service consumer) due to its physical characteristics and characteristics of operation. The WS-POS service consumer must call the **Claim** method and enable exclusive access before most of the methods, properties, and events become effective. If a method is called or a property is set before the exclusive access right is acquired, an illegal error occurs.

If two closely related WS-POS service consumers would like to use an exclusive use device by sharing, there is a way where one WS-POS service consumer has exclusive access to the device for a certain period of time only and releases the exclusive access right later in order to allow the other WS-POS service consumer to also use the device.

If the **Claim** method is called again, the settable characteristics of devices are restored to the state when the **Release** method is called. Examples of restoration include the brightness of the line display, the read track of magnetic stripe readers, and the number of characters per POS printer line. The State characteristics like the sensor property of the POSPrinter are not restored and are set to the current state instead.

2.5.2 Sharable Devices

Some devices are sharable devices. One example is key lock. For sharable devices, the properties can be accessed by the calling of its methods by multiple applications (WS-POS service consumers). In addition, events are notified to all applications to which the device is open. However, a WS-POS service consumer with exclusive access to the sharable device can limit the access to part of the methods and properties and only this WS-POS service consumer can be notified of events.

2.6 Event Messages

The event notifies the application (WS-POS service consumer) of various behavior and changes to the device or removal of the device. There are the following five events.

Event	Description
DataEvent	Input data is stored in device class unique properties.
ErrorEvent	An error occurred during an event-driven input or asynchronous output.
StatusUpdateEvent	State change of the device is reported.
OutputCompleteEvent	Asynchronous output was normally completed.
DirectIOEvent	This event is defined by a service provider and items that cannot be covered by this specification will be provided.

The WS-POS service provider queues events when events are generated. Queued events are notified (delivered) to the application (WS-POS service consumer) when they are ready to be notified. The causes of delays in the sending and receiving of events include the following.

- The WS-POS service consumer has set the **FreezeEvents** property to true.
- The event type is a **DataEvent** or input **ErrorEvent**, and the **DataEventEnabled** property is false.

For terminology concerning the notification of events, the following terms except for "Event" in this paragraph are not used separately, and are published for reference.

Reference: The following terms concerning events are used in this specification.

Queue	When a service decides the necessity of event notification (Fire) to the WS-POS service consumer, the service packs the event in an internal event queue.
Deliver	When the event queue is not empty and all requirements of the first event in the queue are met, this event is removed from the queue and the event notification request for the application is executed.
Fire	It can be said that Fire is the combination of queuing (Queue) and notification (Delivery). However, Fire sometimes is used in broad terms and refers to only one of these steps. The meaning must be identified by the context.

Regulations concerning the management of the event queues are as follows.

- While the device is in the “enable” state, the WS-POS service provider is only queuing (Queue) new events.
- There is a possibility that the WS-POS service provider conducts notification (Delivery) of queued events until the WS-POS service consumer calls the **Release** method (exclusive use device) or the **Close** method (all devices). Note that all remaining events are deleted when either the **Release** or the **Close** methods are executed.
- In the input device, the **ClearInput** method clears data and the input error event. Within the event handler, the WS-POS service consumer can access properties and call methods. However, the WS-POS service consumer must not call the **Release** method and the **Close** method within the event handler since the **Release** method must close event handling (including the thread notifying (Delivering) the event) and **Close** must close the event handling before it returns.

2.7 Input Model

The WS-POS input model supports event-driven input. In event-driven input, input data can be received after **DeviceEnabled** is set to true. The received data is queued as **DataEvent** events for notification to the application (WS-POS service consumer) when the requirements are met. If the **AutoDisable** property is true when data is received, the WS-POS service provider sets **DeviceEnabled** to false and automatically disables the device. As a result, further queuing of input by the service is prevented and the device is physically disabled (if possible).

When the WS-POS service consumer is ready to receive input from the device, the WS-POS service consumer sets the **DataEventEnabled** property to true. When the input is received, the WS-POS service provider queues and notifies the **DataEvent** event. (If the input is already queued, the **DataEvent** event is notified.) This event may include input status information. Immediately before the event is notified, the WS-POS service provider stores input data and other information in the device unique property based on necessity.

The WS-POS service provider makes further data events impossible by internally setting the **DataEventEnabled** property to false immediately before the event is notified. As a result, any subsequent input data is queued by the service while the WS-POS service consumer is processing the current input and related properties. When the WS-POS service consumer completes processing of the current input and is prepared to receive any subsequent data, it sets **DataEventEnabled** to true to restart the notification of events.

If the input device is an exclusive use device, the WS-POS service consumer must both acquire and enable the exclusive access right of the device prior to start of input reading by the device.

In case of sharable input devices, the WS-POS service consumer must open and enable the device prior to the start of input reading by the device. The WS-POS service consumer must call the **Claim** method and request exclusive access to the device before the WS-POS service provider sends data using the **DataEvent** event. When event-driven input is received, if the WS-POS service consumer does not have the exclusive access right to the device, the input is queued until the exclusive access right of the device is acquired (and the **DataEventEnabled** property becomes true). As a result, multiple WS-POS service consumers can share the device in order and input focus can be given and received effectively between WS-POS service consumers.

If the WS-POS service provider finds an error during the input processing of data from event-driven input, the WS-POS service provider changes its **State** property to Error and queues one to two **ErrorEvent** events in order to warn the WS-POS service consumer of the error state. Since this event is not notified until the **DataEventEnabled** property becomes true, the WS-POS service consumer can conduct processing in order. Error events are notified accompanied by data showing the next error position.

InputData:

InputData is queued if an error occurs while one or more **DataEvent** events are queued. This event is queued before all other **DataEvent** events. Due to this event, the WS-POS service consumer can clear the input immediately or has the option of warning users of the error and processing the buffered input data.

The latter case is effective for scanners. Since error warnings can be quickly sent to users, subsequent items are not be scanned until the error is removed. Previously scanned items can be normally processed before error recovery is executed.

Input:

Input is notified when an error occurs and there is no usable data. (In general implementations, this event is set at the end of the event queue.) If part of the input data has already been queued when the error occurs, the **ErrorEvent** event whose error position is **InputData** is

first queued and notified. This error event is notified after all **DataEvent** events are notified. (When the "InputData" event is notified and the event handler of WS-POS service consumer responds "Clear", this "Input" event is not notified.)

In any of the following conditions, the WS-POS service provider ends the Error state when the WS-POS service consumer:

- “Reads” the Input data associated with the InputData **ErrorEvent**.
- “Responds” to the **InputData ErrorEvent** event with an **ErrorResponse** of “Clear”.
- “Calls the **ClearInput** method.

The method for starting an event-driven input is dependent upon the WS-POS service consumer making an input data request. Subsequent input data is not accepted (normally) until the method is issued again. This will re-initialize the input buffer after the WS-POS service consumer has received the previous input data from the device. Examples include the operation of reading MICR data and signature capture data. No new data will be sent from these devices until the service consumer is ready to accept new data. This type of event-driven input is defined as "Asynchronous input".

The **DataCount** property is used to keep track of the number of **DataEvent** events enqueued by the WS-POS service provider.

All input data that is queued by the WS-POS service provider can be deleted by calling the **ClearInput** method. The **ClearInput** method can be called after **Open** in sharable devices and after **Claim** in exclusive use devices.

For the general (Asynchronous) event-driven input model, specific methods, device classes and properties are not required to be defined to return the input data. There are some devices that require methods to return input data and populate specific properties. An example of this is the key lock device. Its locked or unlocked key position status is held in a defined property value determined by an input request. This type of an input is referred to as “Synchronous input”.

2.8 Output Model

The output of WS-POS consists of two output types of *Synchronous* and *Asynchronous*. Each device class may support either, both or neither of these output types.

2.8.1 Synchronous Output

This output type is effective for when it is necessary to output data to a POS device promptly and the application cannot continue until the data has been transferred. The advantage of synchronous output lies in its simplicity.

The application (WS-POS service consumer) calls a class unique method which handles the data output. The WS-POS service provider does not return control back to the application until all the output data has been transferred or an error condition exists.

2.8.2 Asynchronous Output

If the POS device can only handle transfer of data at a low-speed or large blocks of data are required to be transferred, this output type is effective. The application (WS-POS service consumer) sets up a buffer of data that needs to be output to the POS device, executes an asynchronous method call, returns to continue with other computing tasks, and receives a completion or error event when the data has been transferred. Since the WS-POS service consumer can execute other processing when the device is executing the output, the advantage of this output type lies in the ability of the application to multi-task, potentially speeding up POS services.

Further details to the process are as follows:

- The WS-POS service consumer calls a class unique method and starts output. The service buffers the request within program memory, and the service sends it to the physical device immediately as soon as reception processing becomes possible in the physical device. Then, the service sets the identifier of this request in the **OutputId** property and returns back to the application as soon as possible. When the device normally completes the request, the WS-POS service provider notifies the application using an **OutputCompleteEvent** event. The parameter of this event is the **OutputId** of the last successful output complete request.
- If an error occurs when an asynchronous data output is being executed, the **ErrorEvent** event is fired. The event handler of the WS-POS service consumer can retry or clear the output of any outstanding transfer request of queued data. The service remains in the Error state during processing of the **ErrorEvent** event. (Note: If the cause of the error has not been removed when the service returns from an **ErrorEvent** event, another **ErrorEvent** event will fire if the POS device is still in an error state.)

- Asynchronous output is queued based on FIFO buffer technique. All buffered output (including all asynchronous outputs) can be deleted by calling the **ClearOutput** method. The **OutputCompleteEvent** event is not generated as a result of calling the **ClearOutput** method. Calling the **ClearOutput** method should terminate all remaining data from being sent, if possible.

2.9 Device Power Reporting Model

In an application (WS-POS service consumer), the state of the POS device power must be available for a query. This information is conveyed by the **PowerState** enumeration.

Note: *This model does not target notification of the power supply state ("battery on" and "battery low ", etc.) of PCs and the POS terminal main power unit. Notification of this information is managed by utilizing the UnifiedPOS "POS Power" device specification.*

2.9.1 Model

The power state of WS-POS devices may be denoted using one of the following conditions.

- **Online** The POS device is in the “*Power On and Ready*” mode. This means that it is available for use.
- **Off** The POS device is in the “*Power Off*” mode or it is not connected to the POS system. It implies that the POS Device is not available for use.
- **Offline** The POS device is in the “*Not Ready*” mode or cannot respond even though the POS device is in the “*Power On*” mode. It implies that the POS Device is not available for use.
It may be necessary to push a physical button to place the POS device into an “*Online*” mode.
While in the “*Offline*” state the POS device may not be able to respond to requests from an application.
- **OffOffline** The POS device is in an undeterminable “*Off*” or “*Offline*” state. This means the POS device is in a “*Power Off*” or “*Not-Ready*” mode; the device service is not able to distinguish between these two states.

The power state notification only functions for exclusive POS devices if the application is in the programmatic “*open, claim, enable*” condition.

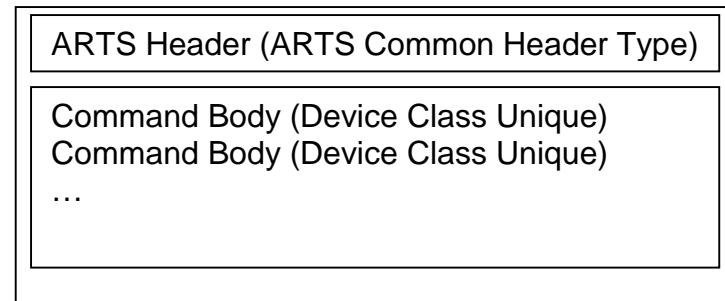
Note: Programmatic vs. Physical (Hardware) “Enable/Disable” State

The context of where the terms “enable/disable” determines the meaning and usually is completely different. In WS-POS application service definition, “enable” or “disable” is a programmatic logical state. In the WS-POS Power Supply Notification model, physical (Hardware) context, “enable” or “disable” refers to the physical state of the POS device. Depending on the POS device, even if the programmatic state is logically “enable”, the physical (Hardware) may be in the “disable” state, physically “offline”. Conversely, if the programmatic state is logically “disable”, the physical (Hardware) state may be Online (“enable”) because it is in the power on and ready mode.

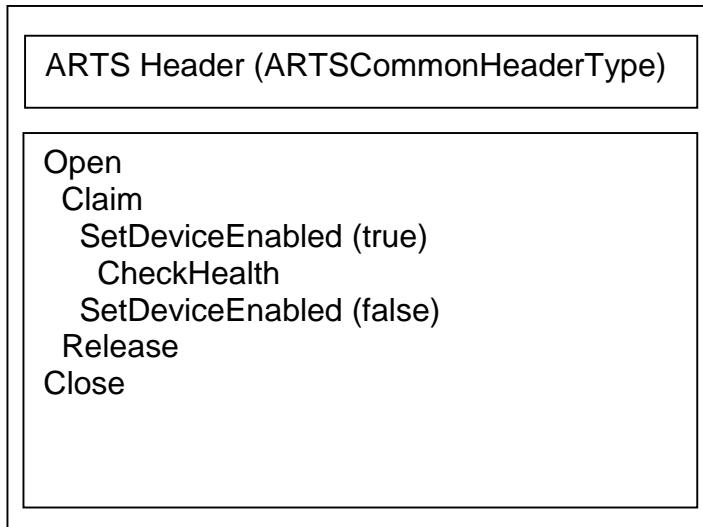
Regardless of the physical (Hardware) state, WS-POS Power State Notification is only available to the application when in the programmatic “enable” state. This restriction is necessary since services can generally only communicate with devices during a programmatic “enable state”. Even when a POS device is physically “Offline”, the service may try and fail a programmatic device “enable”. However, once the POS device becomes “Online” (physical “enable”) and the programmatic state is “enable”, the service will not automatically change the state of the programmatic “enable” to “disable” even if the power state changes.

2.10 ARTS XMLPOS Command Set

In ARTS XMLPOS, the Command Set to transmit a series of processing for methods and properties by batch message is defined. Command Set is comprised of the following.



For example, in the batch processing of **Open, Claim, Enabled, CheckHealth, Disable, Release and Close**,



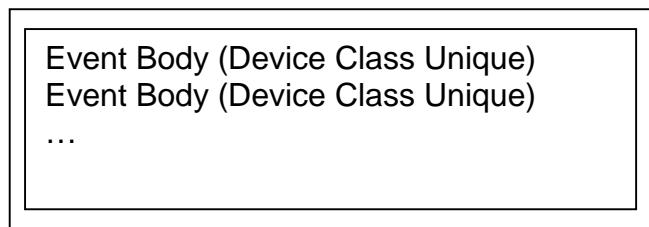
it is possible to batch up the commands and transmit them in one data exchange.

As a result, the message granularity between WS-POS service consumers and WS-POS service providers is flexible and can be appropriately adjusted as the circumstances require.

2.11 ARTS XMLPOS Event Set

In ARTS XMLPOS, Event Set to notify the service consumer of multiple kinds of events collectively is defined.

Event Set is comprised of the following.



2.12 ARTS XMLPOS Schema

WS-POS uses the ARTS XMLPOS schema for the definition of a device class. The ARTS XMLPOS schema is provided in an “xsd” file.

The corresponding schema file names for device classes are as follows.

Schema	Description
<Device Class Name> <Version>.xsd	Schema defining the properties/methods of a device class
<Device Class Name> CommandSet <Version>.xsd	Schema defining Command Set of a device class
<Device Class Name> Event <Version>.xsd	Schema defining the events of a device class
<Device Class Name> EventSet <Version>.xsd	Schema defining Event Set of a device class

Note: In this example, the Version is a character string that starts with prefix “V” and consists of the “major version”, the “minor version”, and the “bug fix” version using period “.” delimiters. (Example: V1.13.1)

Please refer to ARTS SOA Best Practices Technical Report for further version information.

The ARTS XMLPOS schema adopts the version of UnifiedPOS.

2.13 WS-POS WSDL

The WSDL file group below defines the WSDL of the WS-POS service in accordance with the ARTS XMLPOS schema. The WS-POS WSDL's are based upon the corresponding version of the WS-POS standard and the ARTS XMLPOS as defined in the corresponding version of ARTS UnifiedPOS standard.

Although the following WSDL are compatible with the ARTS XMLPOS schema, since binding information is also described in WSDL, the binding information must be changed in accordance with that used by the WS-POS service provider for actual use. The binding information of WSDL included in this WSDL file is the placeholder.

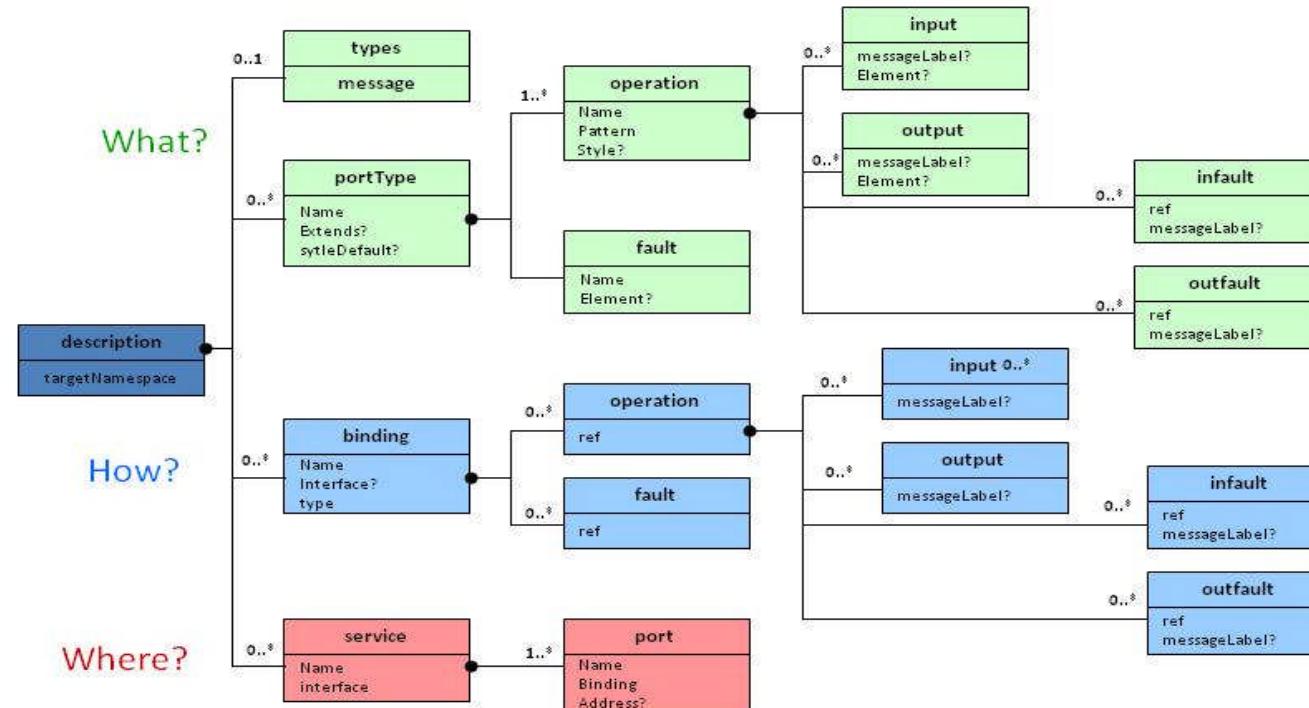
WSDL	Description
<Device Class Name> <Version>.wsdl	WSDL describing the properties /methods of the device class
<Device Class Name> CommandSet <Version>.wsdl	WSDL describing Command Set of the device class
<Device Class Name> Event <Version>.wsdl	WSDL describing the events of the device class
<Device Class Name> EventSet <Version>.wsdl	WSDL describing Event Set of the device class

Note: In this example, the Version is a character string that starts with prefix “V” and consists of the “major version”, the “minor version”, and the “bug fix” version using period “.” delimiters. (Example: V1.13.1)

Please refer to ARTS SOA Best Practices Technical Report for further version information.

The ARTS XMLPOS schema adopts the version of Unified POS.

2.13.1 Web Service Description Language (WSDL)



1

Figure 4: WSDL Overview

WS-POS employs WSDL 1.1, as explained in WS-I basic profile version 1.2, as the core service description facility.

Id	Name	Description
SD001	WSDL	A device's web services MUST be conformant to WSDLs as specified in the "Service Description" section of this specification
SD002	WSA-Action	To be conformant, the receiving Web Services MUST discard SOAP messages that do not contain WS-Addressing Action headers

2.13.2 WS-POS WSDL Interoperability and Development Platform Requirements to Ensure Equivalent Code Conversion

In order to ensure hardware and software interoperability, a WS-POS WSDL must be identical whether it was created from Java code or .NET code.

In practice this means a WCF contract is used to process Windows .NET code to produce a WS-POS WSDL. Also a JAX-WS contract is used to process Java code to create a WS-POS WSDL. Both of these WS-POS WSDLs must be identical. This is the only way of ensuring that WS-POS interoperability can be guaranteed regardless of the development language used to create the WSDL.

The following diagram graphically shows this relationship and the importance of equivalent WS-POS WSDL creation.

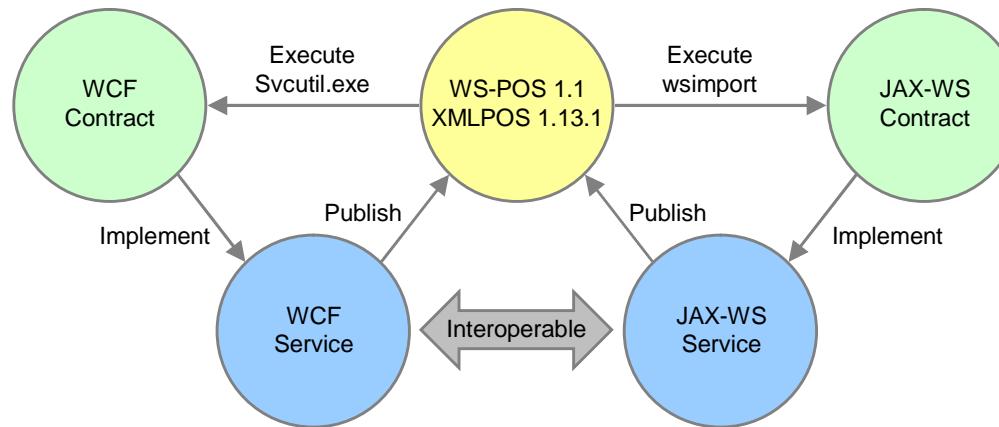


Figure 5: Interoperability and Platform Equivalence to the Code Conversion

The WS-POS 1.1 standard uses the current UnifiedPOS standard XMLPOS messages to define UnifiedPOS classes that are mapped using the ARTS defined WSDLs. This ensures consistent WSDL mapping of the UnifiedPOS Properties, Methods, and Events for the peripheral devices. In addition, the UnifiedPOS Device list of arguments, return values and exceptions are respectively mapped by the WSDL to the Request message, Response message, and Fault message. Finally, the UnifiedPOS data types are mapped to equivalent XML schema data types. Figure 6 shows the relationships of these mappings.

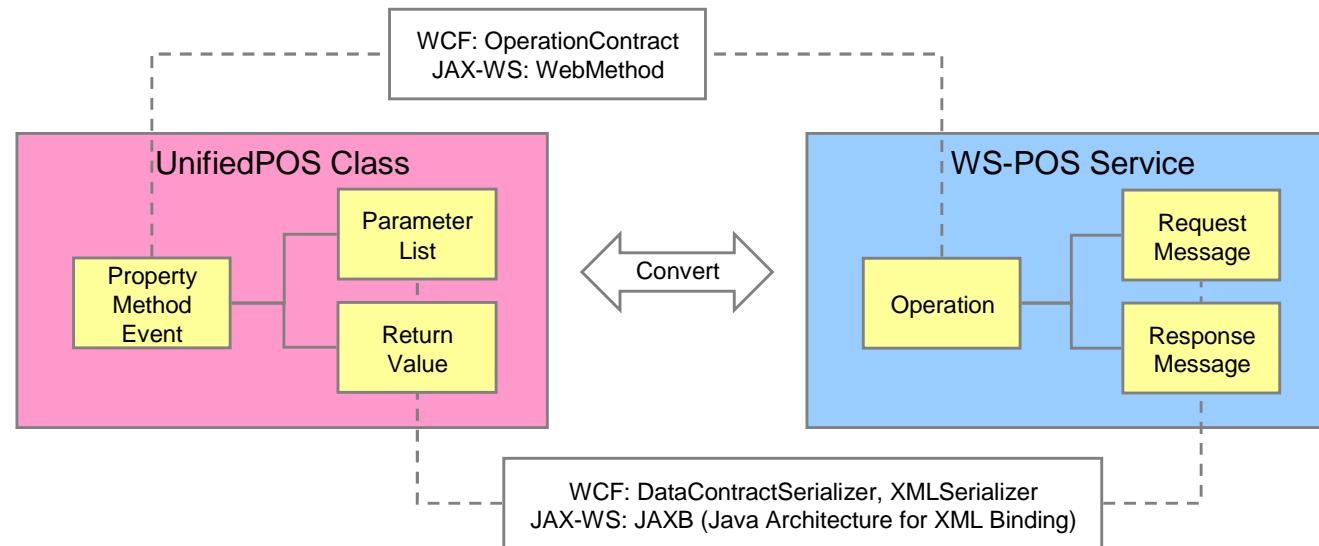


Figure 6: UnifiedPOS and WSDL Mapping

2.13.3 WSDL Documents Provided By ARTS

For each device type, ARTS provides two categories of WSDLs: one to describe the services that a producing device implements, and another to describe the services a consuming application for that device type needs to implement. Each of these provided WSDLs will contain one or more port types with sets of operations, all the messages for each operation, and a corresponding SOAP binding for each port type.

The ARTS WSDL files are part of the WS-POS 1.1 download package. They most recent WS-POS documentation and WSDL package zip and tar files can be found under the WS-POS section at WWW.NRF-ARTS.ORG.

2.13.4 WSDL Documents to Be Created by Implementers

For each device or client wishing to implement WS-POS, it is necessary to create a new WSDL document that imports the corresponding ARTS WSDL. The implementer then creates a service with an endpoint address and ports referring to one or more of the bindings in the ARTS WSDL.

Id	Name	Description
SD003	ARTS-Binding	An implementing service must refer to the bindings from the corresponding provided ARTS WSDL

2.14 Discovery

2.14.1 Description

The ability to find and work with peripheral devices on the LAN (or non-LAN devices represented by proxies on the LAN) is a key requirement for WS-POS. Thus, discovering existing services (or new services after they are introduced) is an important part of the overall solution.

2.14.2 Universal Description, Discovery, and Integration

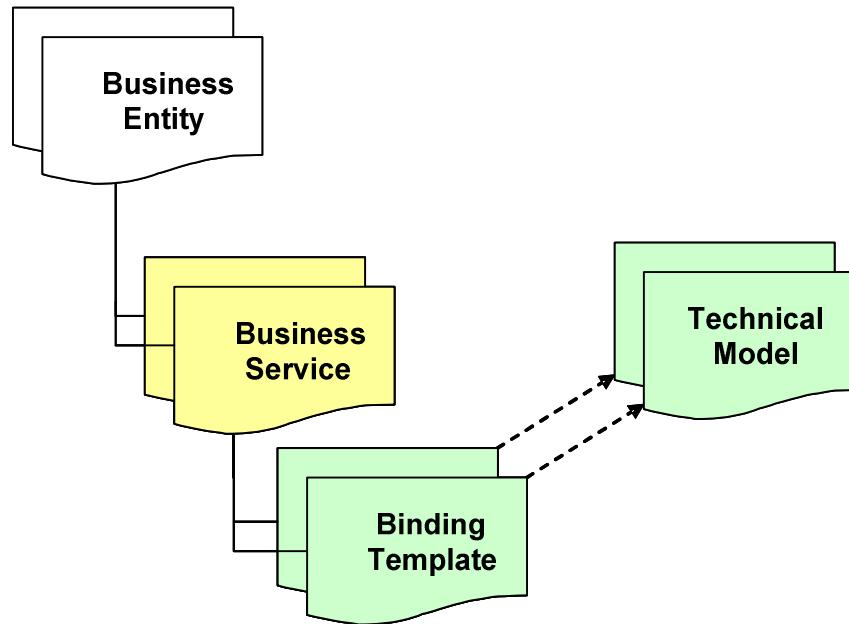


Figure 7: UDDI Overview

WS-POS uses version 3.0.2 of the Universal Description Discovery and Integration (UDDI) specification. A UDDI registry runs on a server and provides a single place for web services in a network to register, and to discover other web services. Version 3.x reduces much of the network chatter associated with previous versions of UDDI.

Id	Name	Description
DI001	UDDI	An implementation MUST be conformant to UDDI version 3.0.2

2.14.3 Registering a device service with UDDI

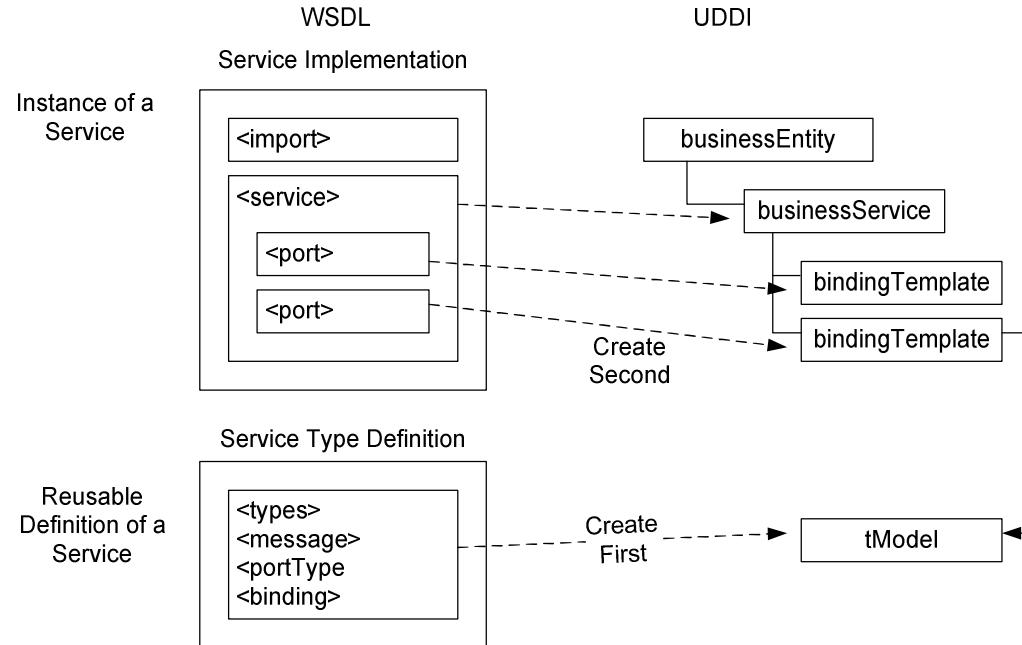


Figure 8: Registering a WSDL with a UDDI Registry

The device services register with UDDI using the mapping scheme described in the technical note from OASIS “Using WSDL in a UDDI Registry”, Version 2.0.2.

2.14.4 Searching for a Service with UDDI

In order to locate a service, all that is required is a search of the UDDI for the business entities that contain the desired services as described in the table below.

ARTS Device Type (businessEntity)	User Service (businessService)	Administrative Service (businessService)
Scanner	ScanItems	ScannerAdmin

2.14.5 Removing a device service from UDDI

Device services should make every effort to ensure that they are not registered with UDDI before going out of service.

2.15 Events

No additional mechanism is required for events beyond simply exposing the UnifiedPOS events as web service methods in a manner analogous to the way that UnifiedPOS methods and properties are exposed.

For example, if there is a “Device”Event (*MSREvent*) type that a particular device produces, that device will expose add “Device”EventListener (*MSREventListener*) and remove “Device”EventListener methods, while the “Device”EventListener will expose a “Device”Occurred method that the device will call remotely when it fires a “Device”Event. This mechanism is an extension of the current UnifiedPOS event model. In addition to exposing ‘open’ ‘claim’ and ‘enable’ methods, a device now exposes an event listener registration method.

While most WS-POS operations are simply UnifiedPOS operations exposed as web services, event registration and deregistration in WS-POS are notably different.

First, because these are web service calls, the arguments for event registration and deregistration methods are WS-Addressing EndpointReferences. While this return value should be the same as that of the device service most of the time, this change allows future versions with support for more complex event topologies to be backwards compatible with this version. Figure 9 depicts this interaction.

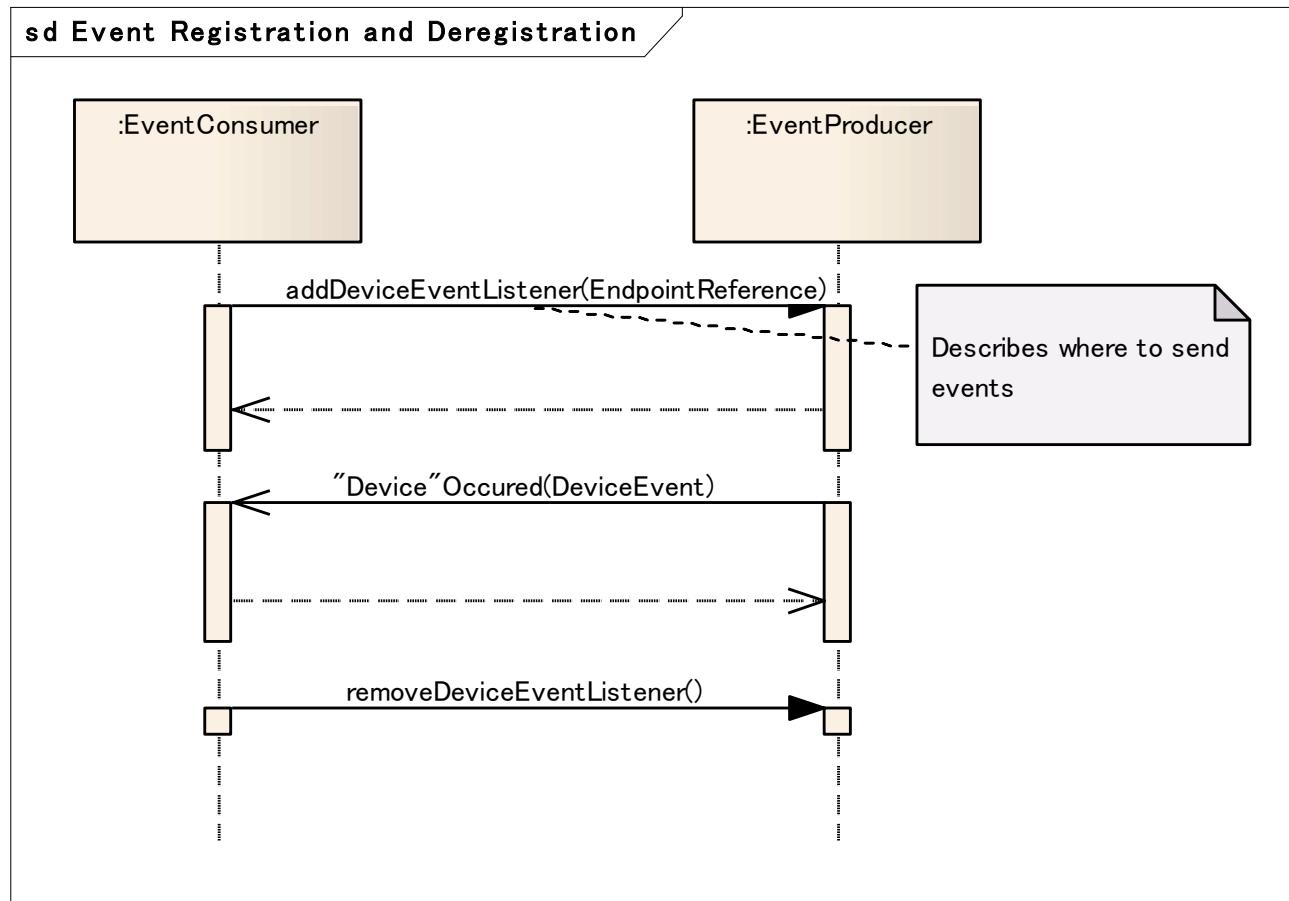


Figure 9: Event Registration and Deregistration

Second, event registration automatically occurs when the **open()** method is called, and deregistration occurs when **close()** is called, as shown in figure 10. Because these registrations happen at different times, the signatures for **open()** and **close()** have changed as well:

```
EndpointReference open(EndpointReference sendEventsTo){ }

void close(EndpointReference stopSendingEventsTo){ }
```

Basically, an open message passes an endpoint reference to an event sink; this method then returns an **EndpointReference** that points to the location to which the consumer sends an unsubscribe message. The close message sends an unsubscribe message to tell the producer to which end point it is to unsubscribe.

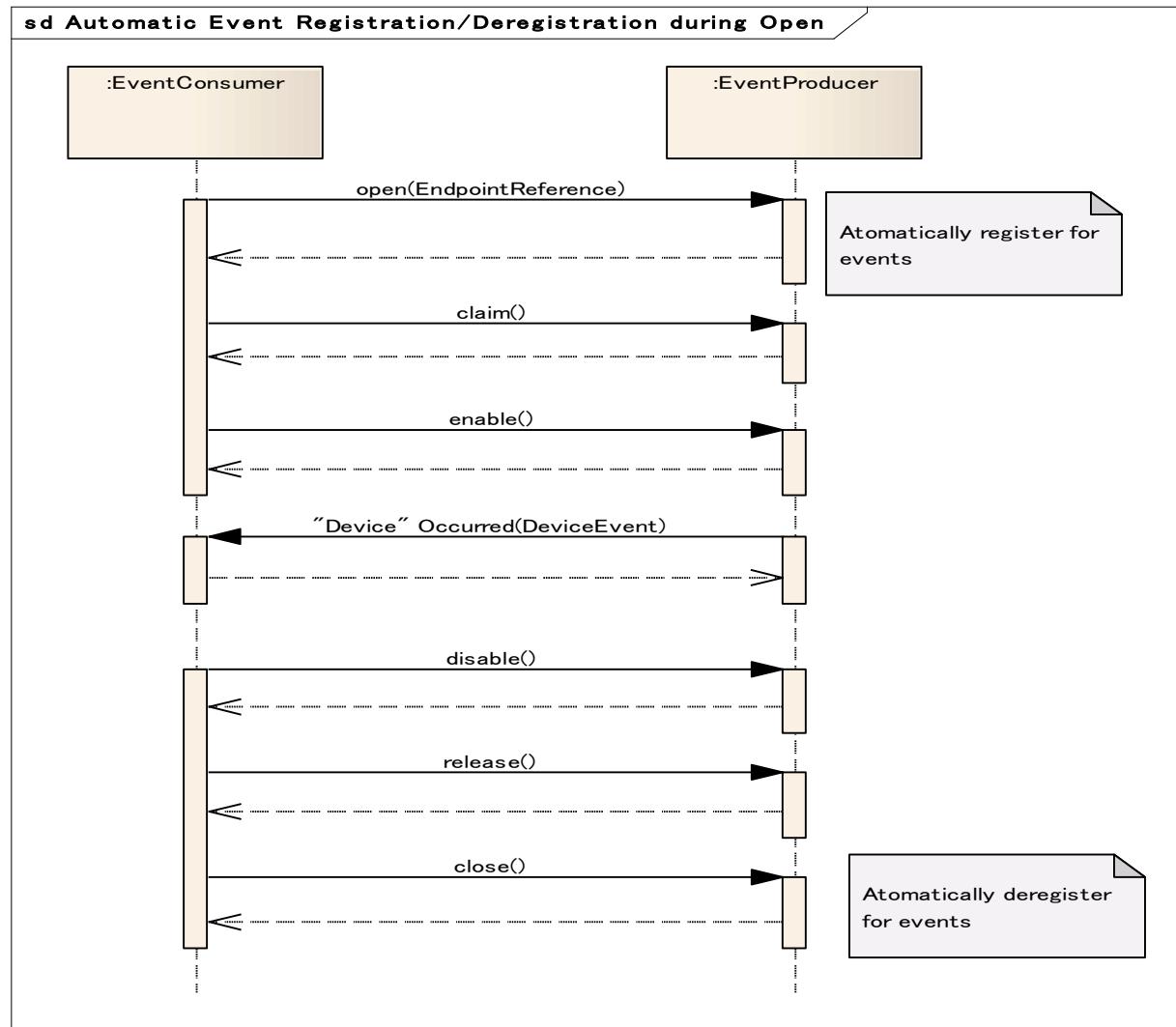


Figure 10: Automatic Event Registration/Deregistration During Open

2.16 Security

2.16.1 Description

Security is a key focus of the entire ARTS distributed peripheral system. In a consumer system of this nature, which exchanges sensitive information such as credit card numbers and details of purchases, it is imperative that the exchanged messages are secure. Measures must be taken to ensure that devices are authenticated, that authorizations are assured, that transmitted information is always safe from snooping and alteration and that the transmitted information goes to the proper endpoint.

2.16.2 Transport Layer Security

For point to point messages, Transport Layer Security (TLS) 1.2 will be employed, and for message level security, WS-POS will use the WS-Security core, version 1.1.

Id	Name	Description
SE001	TLS	An implementation MUST be conformant to TLS version 1.2

2.16.3 Device Authentication with UDDI

The UDDI server must have a method to verify that device keys are acceptable. This may take the form of a preloaded table, verification with some outside authority, etc.

Devices must provide signatures for their interactions with UDDI. The UDDI server must then use these signatures to validate that the device is allowed to use the server.

Id	Name	Description
SE002	DeviceAuthUDDI	UDDI must reject all unsigned messages from devices
SE003	DeviceAuthUDDIFail	UDDI must reject all messages that are signed by unauthorized devices.

2.16.4 Client Authentication with UDDI

The UDDI server must have a method to verify that client keys are acceptable. This may take the form of a preloaded table, verification from some outside authority, etc.

Clients must provide signatures for their interactions with UDDI. The UDDI server must then use these signatures to validate that the client is allowed to use the server.

Id	Name	Description
SE004	ClientAuthUDDI	UDDI must reject all unsigned messages from clients
SE005	ClientAuthUDDIFail	UDDI must reject all messages that are signed by unauthorized clients.

2.16.5 Client Authentication with Device

When interacting with devices, the client must sign all of its messages. The first message, which opens the device, will contain an endpoint reference that the device will send events to. The device must store an (endpoint reference, signature) name-value pair for the client in order to validate future communications. When the client sends messages to the device, the device will use the endpoint reference in the “<from>” WS-Addressing field along with the included signature to validate the message. When opening and closing the device, the device must validate that the endpoint reference in the “<from>” WS-Addressing field matches the endpoint reference in the SOAP-Body.

Id	Name	Description
SE006	ClientAuthDevice	Device must reject all unsigned messages from clients
SE007	FromSIGMatch	Device must reject all messages whose <from> element in the header does not match the EndpointReference stored for that signature
SE008	FromEPRMatch	Device must reject all open and close messages in which the EndpointReference in the SOAP body's <in> doesn't match the <from> in the SOAP header

Note: A client only has the ability to register its own endpoint reference; no attempt to define how a client can register any other endpoint references has been made. This is a known endpoint reference registration limitation issue currently in all forms of web services event registration.

2.17 XML Payload

While it is described in the WSDLs, it is worth noting that the XML payload for all the SOAP messages is almost entirely specified using XMLPOS, with the exception of the methods that deal with event registration and deregistration, which are **open()** and **close()**, as well as the callback methods on the clients.

3. GENERAL FLOW

3.1 General WS-POS Flows

This section covers the prescribed methods for using the WS-POS stack.

3.1.1 Device service initializes

1. Setup device - Address, capabilities, naming, location of UDDI
2. Register with UDDI – authenticate itself, address, capabilities, naming
 - a. Message signed with device private key
 - b. UDDI validates signature

3.1.2 Device shutdown

1. Device deletes itself from UDDI
2. Power off scanner device

3.1.3 Client startup

1. Setup client entity service - address, capabilities, naming, location of UDDI

3.1.4 Client interaction with devices

1. Query UDDI for list of device services – authenticate itself
 - a. Message signed using Client private key
2. UDDI returns list of devices to client- Client chooses which device to connect to
3. Client entity service opens device service, registers for events (atomic)
 - a. Message signed with Client private key
4. Device validates signature
5. Device stores endpoint reference/certificate pair for future verification
6. Client entity service – claim, enable device service

- a. both messages signed with Client private key
- 7. User interacts with device
 - a. device service sends events to the client (Signed with Device Private key)
- 8. As client changes settings, settings are stored in endpoint reference/certificate table
 - a. All messages to device are signed by Client
- 9. Client entity service disables, releases device
 - a. Message signed with Client private key
- 10. Client entity service closes device
 - a. Message signed with client private key
 - b. Device validates client endpoint reference against signature

3.2 Simple Use Case

3.2.1 Client Entity Service Acquiring and Utilizing a Scanner Device Service

A client entity service used in a point-of-service solution allows the clerk to scan items. This service acquires and utilizes a scanner device service to input transaction item identification. The client entity service releases the scanner device service after the sales transaction completes.

3.2.1.1 SOA Main Flow Description

1. Setup UDDI – authentication parameters for scanner device service and client entity service
2. Power on the scanner device
3. Scanner device service initializes
 - a. Setup scanner - Address, capabilities, naming, location of UDDI
 - b. Register with UDDI – authenticate itself, provide implementation WSDL
 - i. Message signed with Scanner private key
 - ii. UDDI validates signature
4. The Client entity service is loaded and executed.
 - a. Setup client entity service - address, capabilities, naming, location of UDDI
 - b. Query UDDI for list of scanner device services – authenticate itself
 - i. Message signed using Client private key
5. UDDI returns list of devices - Client entity service chooses which scanner to connect to
6. Client entity service opens scanner device service, registers for events (atomic)
 - a. <?xml version="1.0" encoding="http://schemas.xmlsoap.org/soap/envelope/" standalone="no"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wsa="<http://www.w3.org/2005/08/addressing>" xmlns:ns3="http://www.nrf-arts.org/UnifiedPOS/Scanner/"><soapenv:Header><wsa:To><http://scanner.ws-pos-example.nrf-arts.org:8081/services/Scanner-WS></wsa:To><wsa:MessageID>urn:uuid:C5C856C8175724AD9E1195687902183</wsa:MessageID><wsa:Action><http://www.nrf-arts.org/UnifiedPOS/Scanner/Open></wsa:Action></soapenv:Header>

```
<soapenv:Body>
  <ns3:Open>
    <ns3:EndpointAddress>http://192.168.1.1:8082/axis2/services/Client-WS</ns3:EndpointAddress>
  </ns3:Open>
</soapenv:Body>
</soapenv:Envelope>
```

b. Message signed with Client private key

7. Scanner validates signature

Scanner stores endpoint reference/certificate pair

8. Client entity service – claim, enable scanner device service (both messages signed with Client private key)

```
a. <?xml version="1.0" encoding="http://schemas.xmlsoap.org/soap/envelope/" standalone="no"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wsa="http://www.w3.org/2005/08/addressing"
  xmlns:ns3="http://www.nrf-arts.org/UnifiedPOS/Scanner/">"
  <soapenv:Header>
    <wsa:To>http://scanner.ws-pos-example.nrf-arts.org:8081/services/Scanner-WS</wsa:To>
    <wsa:MessageID>urn:uuid:C5C856C8175724AD9E1195687902183</wsa:MessageID>
    <wsa:Action>http://www.nrf-arts.org/UnifiedPOS/Scanner/Claim</wsa:Action>
  </soapenv:Header>
  <soapenv:Body>
    <ns3:Claim>
      <ns3:Timeout>1000</ns3:Timeout>
    </ns3:Claim>
  </soapenv:Body>
</soapenv:Envelope>

b. <?xml version="1.0" encoding="http://schemas.xmlsoap.org/soap/envelope/" standalone="no"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wsa="http://www.w3.org/2005/08/addressing"
  xmlns:ns3="http://www.nrf-arts.org/UnifiedPOS/Scanner/">"
  <soapenv:Header>
    <wsa:To>http://scanner.ws-pos-example.nrf-arts.org:8081/services/Scanner-WS</wsa:To>
    <wsa:MessageID>urn:uuid:C5C856C8175724AD9E1195687902183</wsa:MessageID>
    <wsa:Action>http://www.nrf-arts.org/UnifiedPOS/Scanner/SetDeviceEnabled</wsa:Action>
  </soapenv:Header>
  <soapenv:Body>
    <ns3:SetDeviceEnabled>
      <ns3:DeviceEnabled>true</ns3:DeviceEnabled>
    </ns3:SetDeviceEnabled>
  </soapenv:Body>
</soapenv:Envelope>
```

9. As client changes settings, settings are stored on a per-client basis

10. Clerk scans an item

11. Scanner device service sends WS-POS event to the client entity service (calling the client entity service callback method using XMLPOS) (Signed with Scanner Private key)

```
<?xml version="1.0" encoding="http://schemas.xmlsoap.org/soap/envelope/" standalone="no"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:ns3="http://www.nrf-arts.org/UnifiedPOS/ScannerEvents/">
    <soapenv:Header>
        <wsa:To>http://client.ws-pos-example.nrf-arts.org/services/Client-WS</wsa:To>
        <wsa:ReplyTo>
            <wsa:Address>http://www.w3.org/2005/08/addressing/none</wsa:Address>
        </wsa:ReplyTo>
        <wsa:MessageID>urn:uuid:548181C361F2A1EE5C1195691857095</wsa:MessageID>
        <wsa:Action>http://www.nrf-arts.org/UnifiedPOS/ScannerEvents/DataEvent</wsa:Action>
    </soapenv:Header>
    <soapenv:Body>
        <ns3:DataEvent>
            <ns3:Source>http://scanner.ws-pos-example.nrf-arts.org/services/Scanner-WS</ns3:Source>
            <ns3:EventID>1234</ns3:EventID>
            <ns3:TimeStamp>2011-01-09T12:34:56</ns3:TimeStamp>
            <ns3>Status>0</ns3>Status>
        </ns3:DataEvent>
    </soapenv:Body>
</soapenv:Envelope>
```

Client entity service receives the event

12. Client entity service disables, releases and closes scanner device

```
<?xml version="1.0" encoding="http://schemas.xmlsoap.org/soap/envelope/" standalone="no"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:ns3="http://www.nrf-arts.org/UnifiedPOS/Scanner/">
    <soapenv:Header>
        <wsa:To>http://scanner.ws-pos-example.nrf-arts.org/services/Scanner-WS</wsa:To>
        <wsa:MessageID>urn:uuid:C5C856C8175724AD9E1195687902183</wsa:MessageID>
        <wsa:Action>http://www.nrf-arts.org/UnifiedPOS/Scanner/SetDeviceEnabled</wsa:Action>
    </soapenv:Header>
    <soapenv:Body>
        <ns3:SetDeviceEnabled>
            <ns3:DeviceEnabled>false</ns3:DeviceEnabled>
        </ns3:SetDeviceEnabled>
    </soapenv:Body>
</soapenv:Envelope>
<?xml version="1.0" encoding="http://schemas.xmlsoap.org/soap/envelope/" standalone="no"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:ns3="http://www.nrf-arts.org/UnifiedPOS/Scanner/">
    <soapenv:Header>
        <wsa:To>http://scanner.ws-pos-example.nrf-arts.org/services/Scanner-WS</wsa:To>
```

```
<wsa:MessageID>urn:uuid:C5C856C8175724AD9E1195687902183</wsa:MessageID>
<wsa:Action>http://www.nrf-arts.org/UnifiedPOS/Scanner/Release</wsa:Action>
</soapenv:Header>
<soapenv:Body>
    <ns3:Release />
</soapenv:Body>
</soapenv:Envelope>
```

13. Client entity service closes scanner, unregisters for events (Signed with Client private key)

```
<?xml version="1.0" encoding="http://schemas.xmlsoap.org/soap/envelope/" standalone="no"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:wsa="http://www.w3.org/2005/08/addressing"
xmlns:ns3="http://www.nrf-arts.org/UnifiedPOS/Scanner/">
    <soapenv:Header>
        <wsa:To>http://scanner.ws-pos-example.nrf-arts.org/services/Scanner-WS</wsa:To>
        <wsa:MessageID>urn:uuid:C5C856C8175724AD9E1195687902183</wsa:MessageID>
        <wsa:Action>http://www.nrf-arts.org/UnifiedPOS/Scanner/Close</wsa:Action>
    </soapenv:Header>
    <soapenv:Body>
        <ns3:Close>
            <ns3:EndpointAddress>http://192.168.1.1:8082/axis2/services/Client-WS</ns3:EndpointAddress>
        </ns3:Close>
    </soapenv:Body>
</soapenv:Envelope>
```

14. Scanner device service notifies UDDI to delete service

15. Power off scanner device

3.3 Use Case Catalog

This section provides example usage for the WS-POS Specification. They were provided by domain experts from the WS-POS-J Initiative Business Scenario Working Group Contributors in Japan. These use cases were left as is in order to preserve the original intended meaning as provided by the international community.

These are meant as examples to aid in the understanding of the WS-POS specification and are not meant as an exhaustive list for usage.

3.4 Scope

Contributor: Seiko Epson Corporation

This section describes a high-level use case that covers new business scenarios based on WS-POS in general. The subsequent use cases are the embodiment or combination of the basic use case described here.

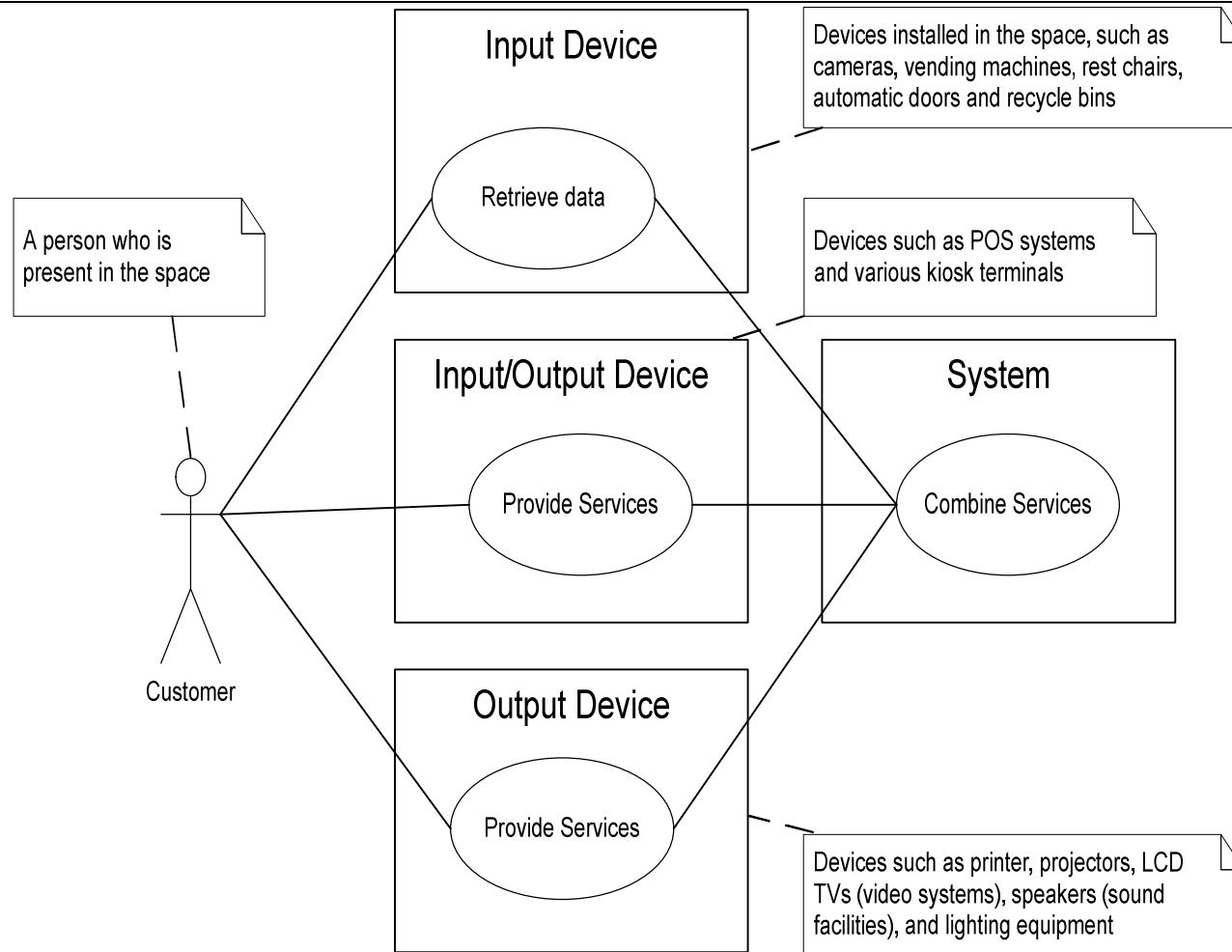
Space where all devices and applications behave as services

For an environment where input devices, output devices and input/output devices placed in every space (including rest areas, common areas, as well as every store in shopping malls) behave as services, the system offers an application program that provides new services by combining various devices to generate space where a wide variety of ideas can suggest new services to customers.

Definitions of entities:

Customer	A person who is present in the space
Input device	Devices installed in the space, such as cameras, vending machines, rest chairs, automatic doors, and recycle bins
Output device	Devices such as printers, projectors, LCD TVs (video systems), speakers (sound facilities), and lighting equipment
Input/output device	Devices such as POS systems and various kiosk terminals
Application service	An application program that provides services by combining the above input, output, and input/output devices

WS-POS 1.1 Technical Specification



WS-POS 1.1 Technical Specification

High-Level Use Case

Use Case –			
Stakeholders		Concerns	
Customer		A customer wants to have a comfortable time in the space. A customer also wants to enjoy satisfactions through delicate and better services.	
Store manager		A store Manager wants to draw more customers and increase sales by providing high-value-added services to customers.	
Device vendor		A device vendor wants to increase sales by providing easy-to-use devices and bringing the possibility of enhancing the added value of devices beyond expectation.	
Software vendor		A software vendor wants to increase sales by providing the new value through flexible combination of various device functions and new suggestions to Store Managers through embodiment of various ideas.	
Actors		Type	Description
Customer			A person who visits the space to do some shopping or kill time.
Input device			In addition to input devices such as surveillance cameras, information of sensors attached to automatic doors may be supposed to be input devices. When sales information of vending machines can be managed in real time, the information can also be used as an input device.
Output device			In addition to output devices such as printers installed on kiosk terminals, video output devices (including projectors and LCD TVs), sound output devices (including speakers), and software-controlled lighting and air conditioning equipment can be used as output devices.
Input/output device			Devices that have both input and output device functions (such as POS terminals and kiosk terminals) can be used both as input devices and output devices.
Application program			When each input, output, input/output device can be used as service equipment, controlling output devices based on information from input devices enables programs to provide a wide variety of services depending on ideas.

WS-POS 1.1 Technical Specification

Major Normal Scenario:

Label	Scenario
1.	A customer inputs information from Input Devices (but not always inputs information consciously)
1.1.	An input Device distributes the input information to Application Programs
1.2.	An application Program analyzes the input information and set up output information
1.3	An application Program sends the output information to Output Devices
1.4	An output Device distributes the output information to Customers
1.5	A customer receives the output information from Output Devices (but not always receives the information consciously)

3.5 Sub Scope: Linkage between In-Store Kiosk and POS (Devices)

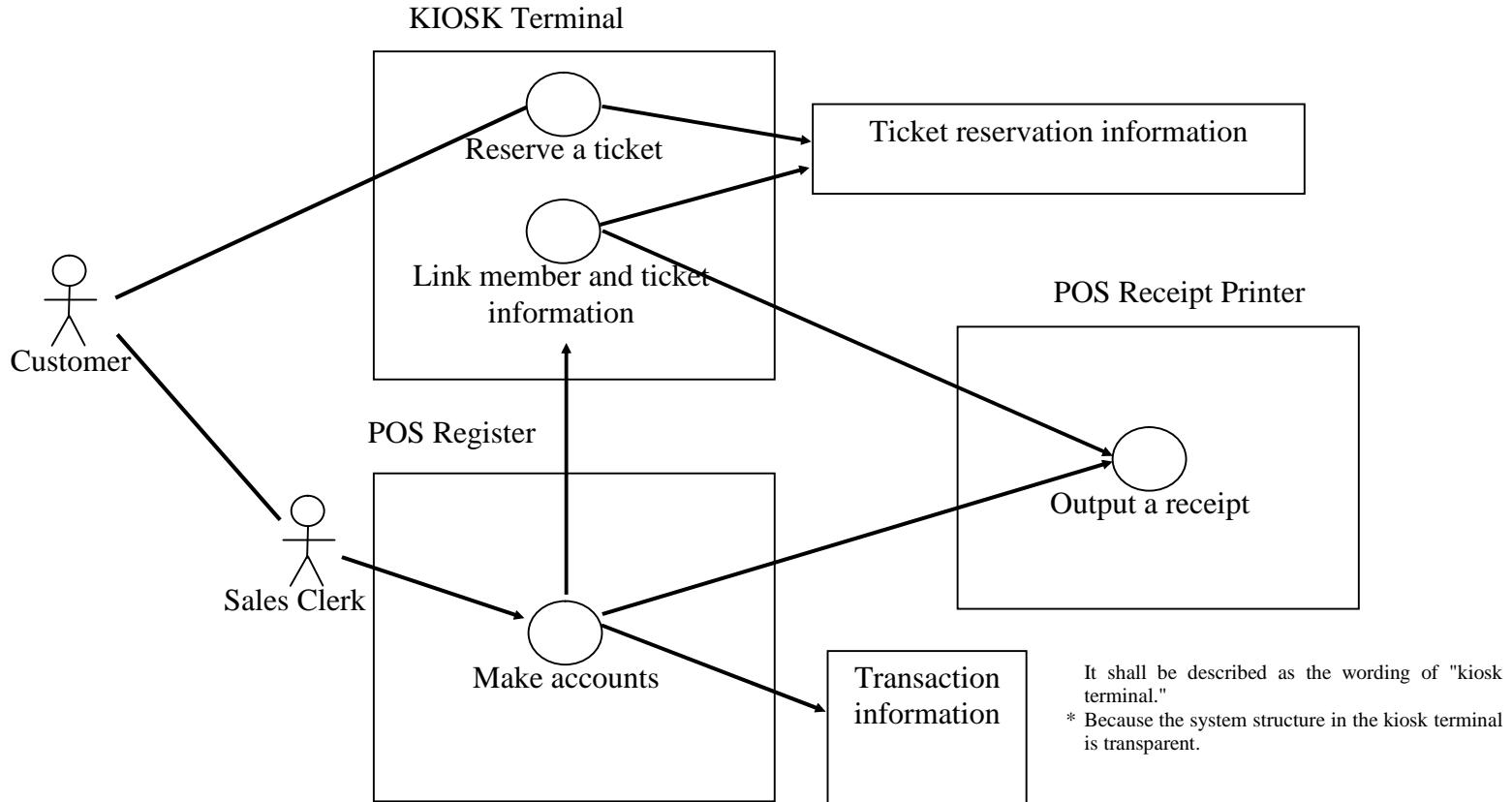
Contributor: Vinculum Japan Corporation

When a customer checks information (such as seat reservation information of a movie theater or an airplane) with an in-store kiosk and reserves a ticket, the system outputs the reservation information from a POS printer on the LAN.

Definitions of entities:

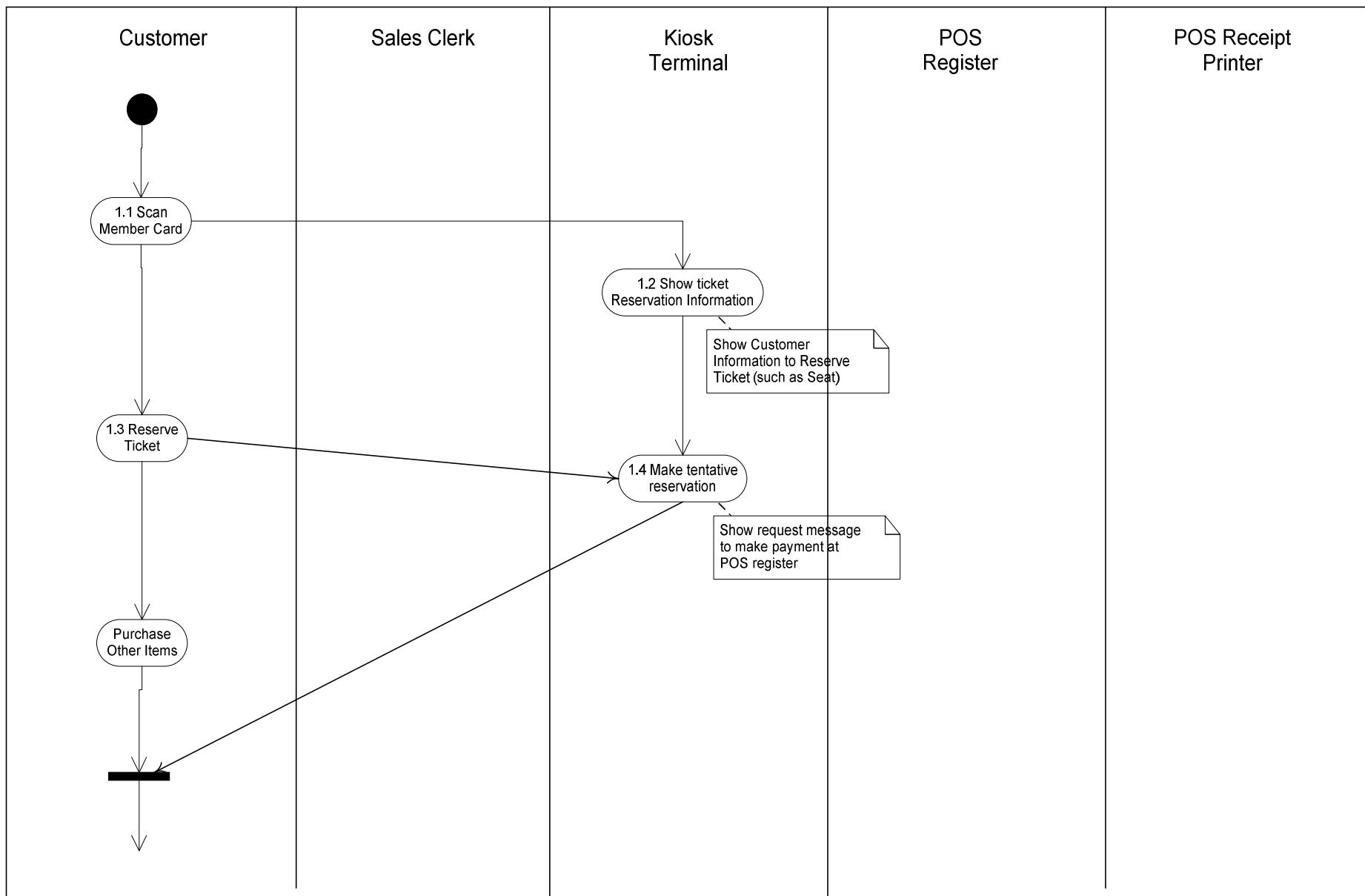
- Customer: A person who reserves a ticket or purchases goods.
- Kiosk terminal: A terminal that manages various information display and ticket reservations.
- POS register: A terminal to make accounts for in-store goods.
- Sales clerk: A person who operates the POS register.
- POS receipt printer: A terminal to output transaction receipts or ticket reservation receipts.
- Ticket reservation information: Information (such as information on seat reservations) managed by the kiosk terminal.
- Transaction information: Actual accounting information managed by the POS register.

WS-POS 1.1 Technical Specification

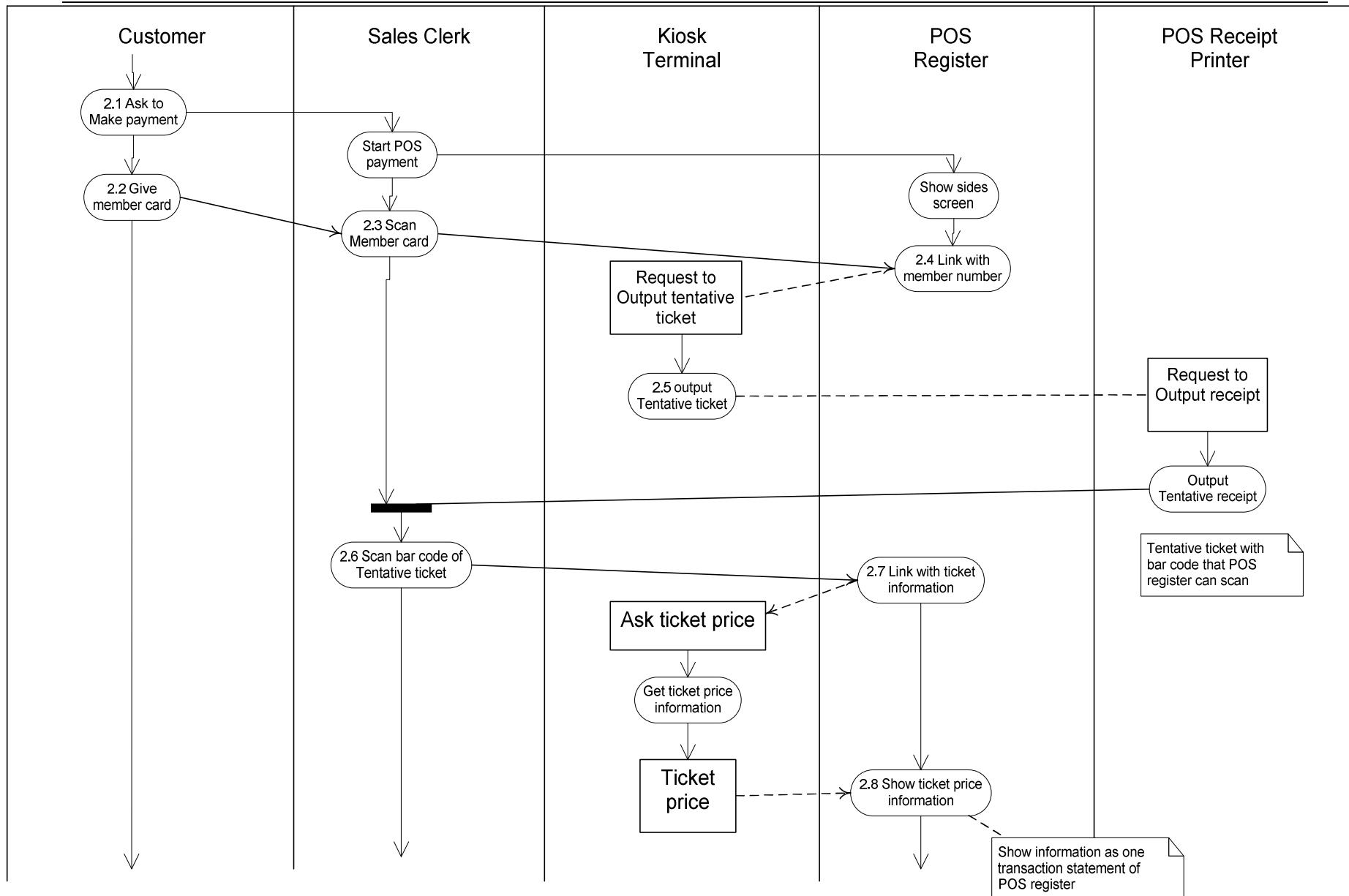


WS-POS 1.1 Technical Specification

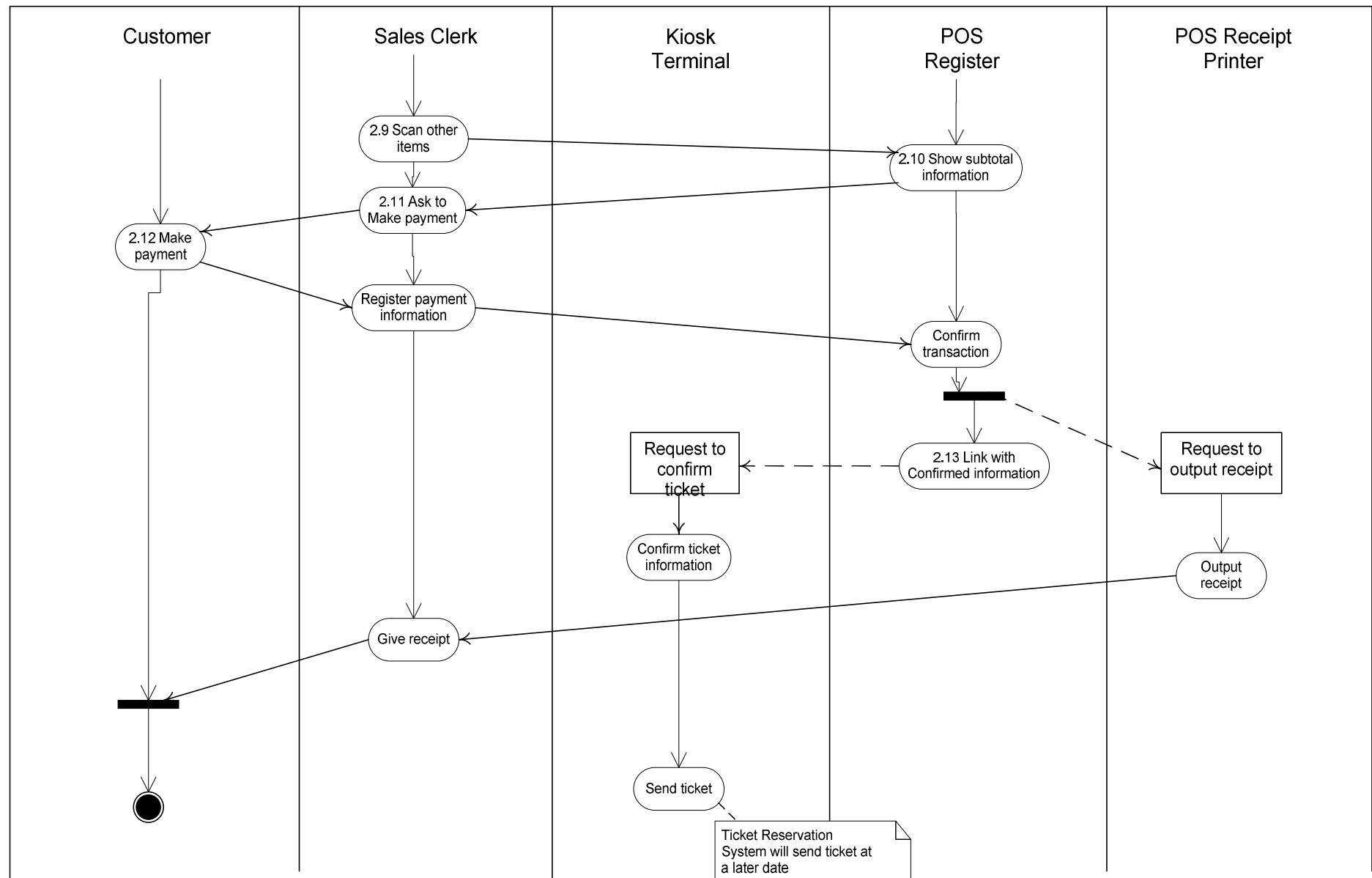
Activity Diagram



WS-POS 1.1 Technical Specification



WS-POS 1.1 Technical Specification



3.6 Sub Scope: Linkage between Sales Assistance Terminals and POS Devices

Contributor: Toshiba Tec Corporation

At apparel retailers, the system that helps sales clerks provide the following services to customers:

- Provide detailed information on items and the availability of items.
- Provide item-related information including arrival schedule of new items, recommended apparel coordinations, and hot-selling items.
- Pre-purchase functions before making payment with POS for customer's purchased items.

This support system consists of the following:

- Backend services (created with Web service applications and others) that provide search/reference functions for item information and purchase process functions,
- Sales assistance terminals carried by every sales clerk to read item codes (bar codes or RFID is used) of items and provide information to customers by collaborating with the backend services,
- POS terminals with a printer to print item information and receipts.

Definition of entities:

- Sales assistance terminal:

Carried by a sales clerk, an in-store information terminal (PDA or tablet PC) for sales clerks to introduce items to customers and search inventory upon customer's request. The terminal is used as a client of the Sales Assistance System. To read item codes, the terminal is equipped with a bar code reader or an RFID reader device.

- Receipt printer:

A printer connected to a POS terminal to print a receipt or an acknowledgment at payment, normally.

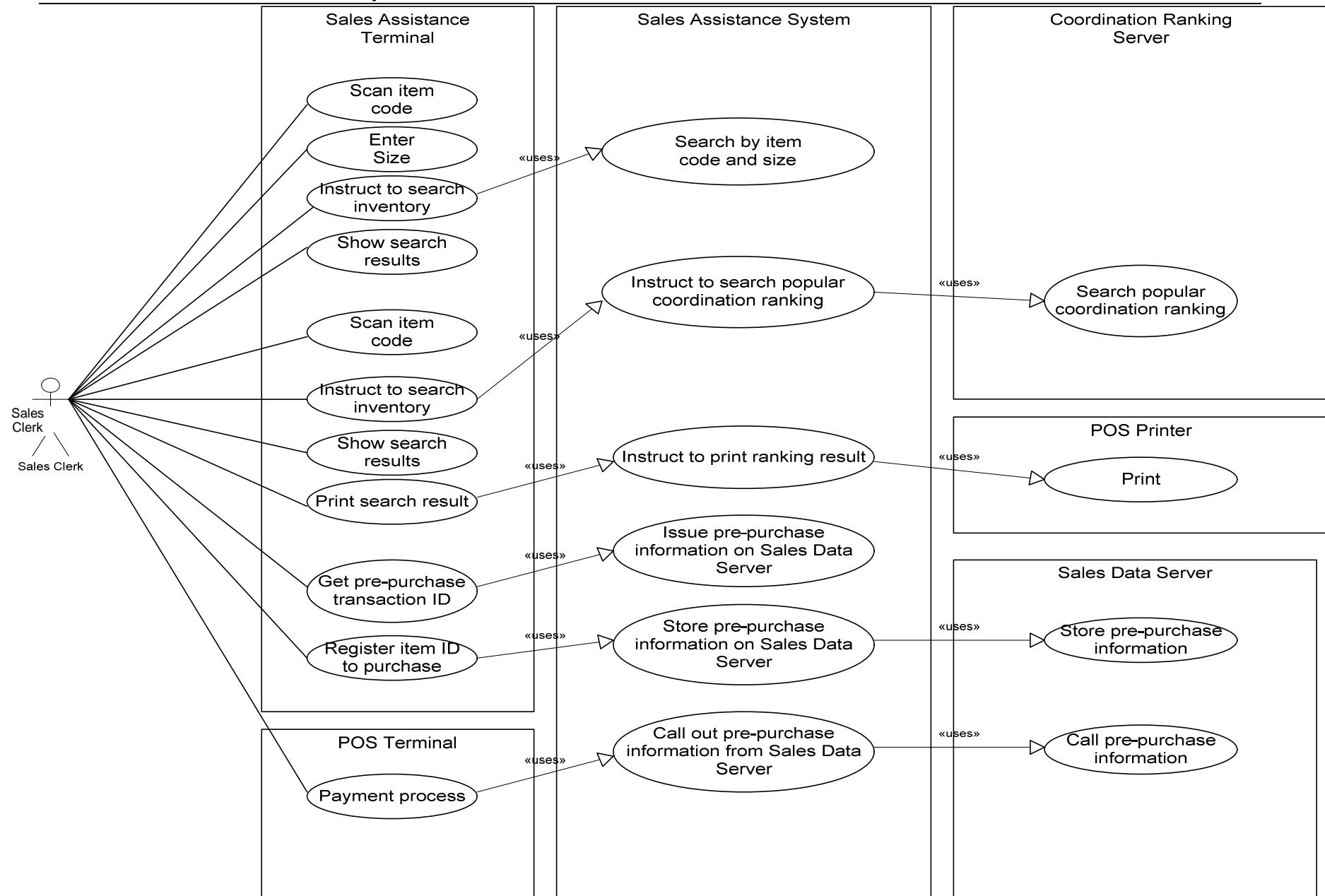
- POS terminal:

WS-POS 1.1 Technical Specification

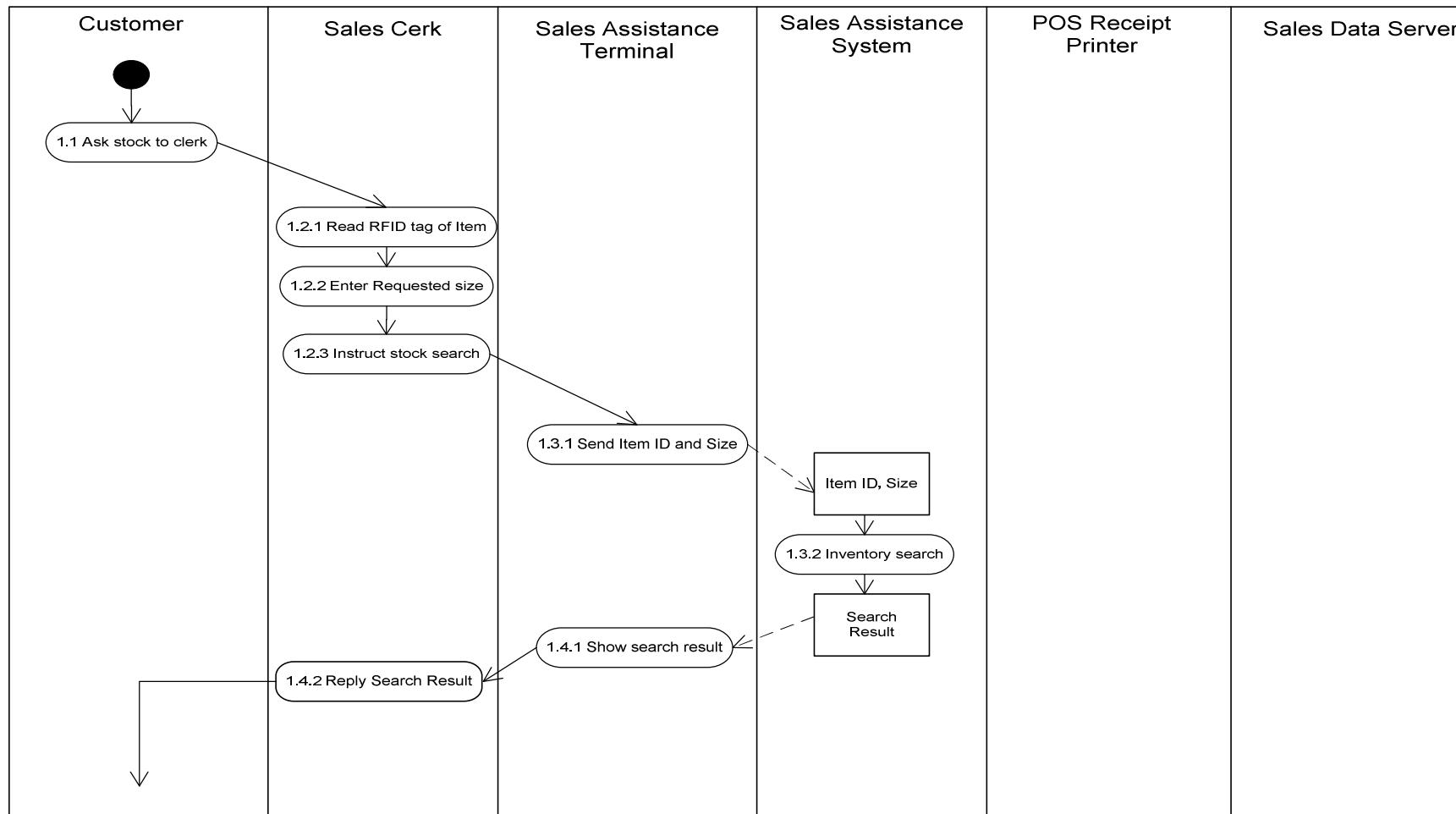
A terminal used by a sales clerk when customers make purchase process. When the clerk registers items given from a customer with the terminal, the terminal displays the total amount of the purchase. Then the customer pays the amount. To handle price and discount information, general POS terminals communicate with servers that contain information on items and prices. Most POS terminals are equipped with a receipt printer, a bar code scanner, price display screens (including a screen for clerks and a screen for customers), and POS keyboard.

- Item:
 - Goods or services that a customer purchases. A customer makes its payment by cash or other means in exchange for the item to purchase.
When seen from the system side, an item has secondary information including ID, name, and price.
- Inventory:
 - Items on the store shelves or stored in backyards or warehouses.
- Item-related information such as coordination rankings
 - Reference information for customers when they purchase an item. Ranking information indicates the latest trend or popularity of coordinations. Customers refer to the information when purchasing items.

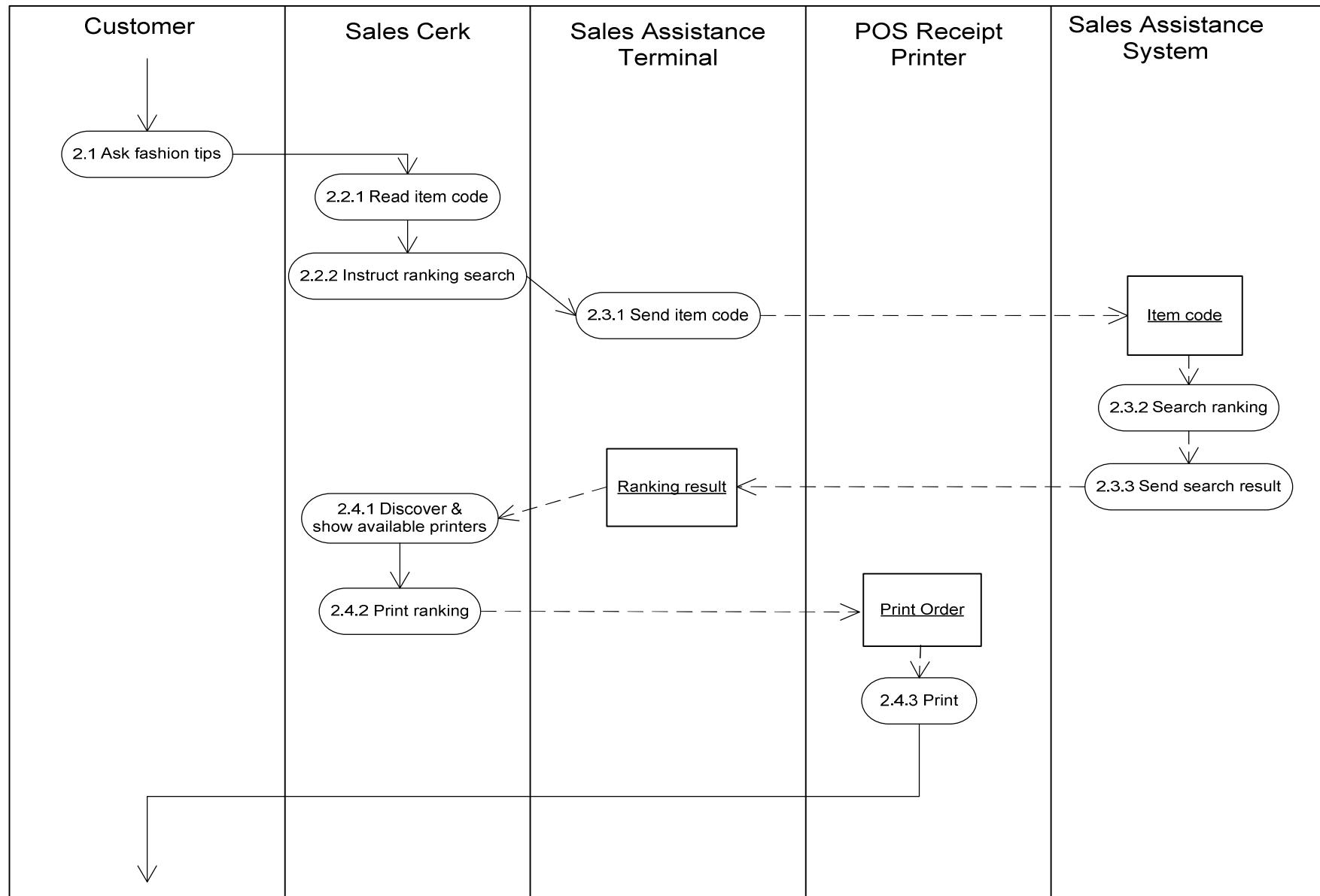
WS-POS 1.1 Technical Specification



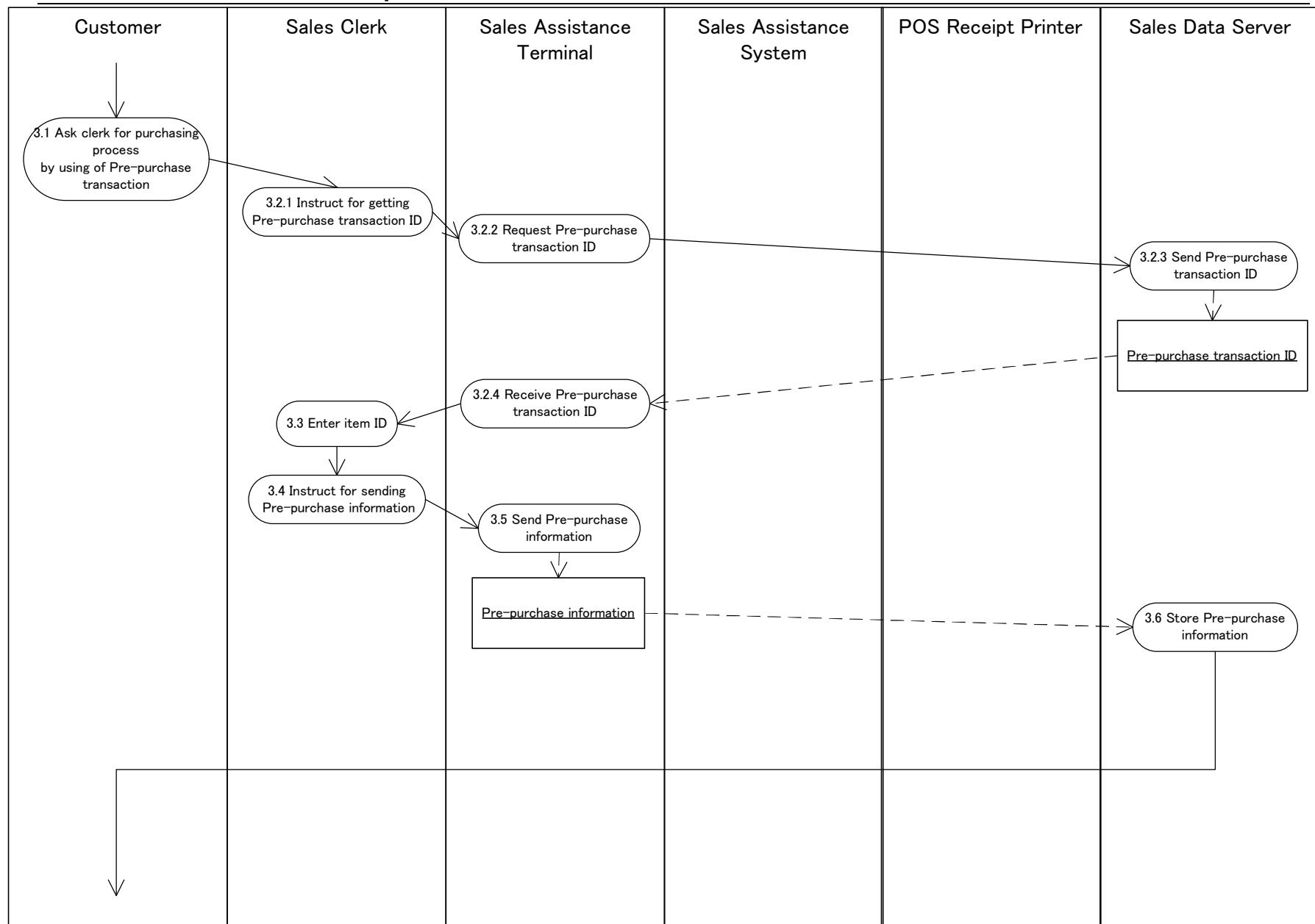
Activity Diagram



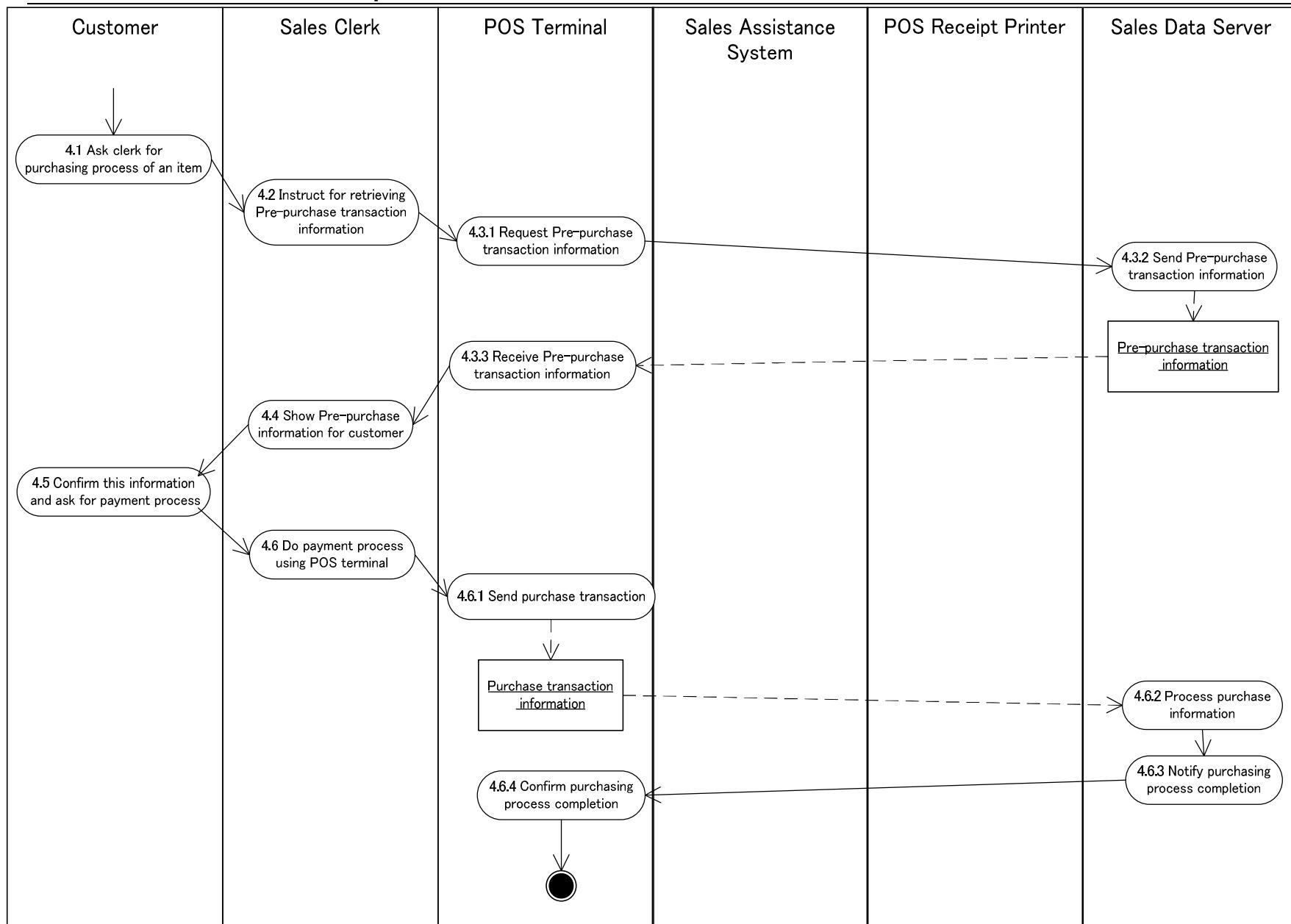
WS-POS 1.1 Technical Specification



WS-POS 1.1 Technical Specification



WS-POS 1.1 Technical Specification



3.7 Sub Scope: Batch Payment in Commercial Complex

Contributor: Sorimachi Giken Co., Ltd.

At a commercial complex such as a department store or a shopping mall, the system makes multiple payments at once at the dedicated payment corner or POS in any retail store instead of making individual payment at the respective store.

This sub scope is independent of whether a customer takes items home or makes them delivered to home.

When delivering, a customer has to write down its address only once at any store. Then the system delivers all of the customer's items purchased in the commercial complex at once.

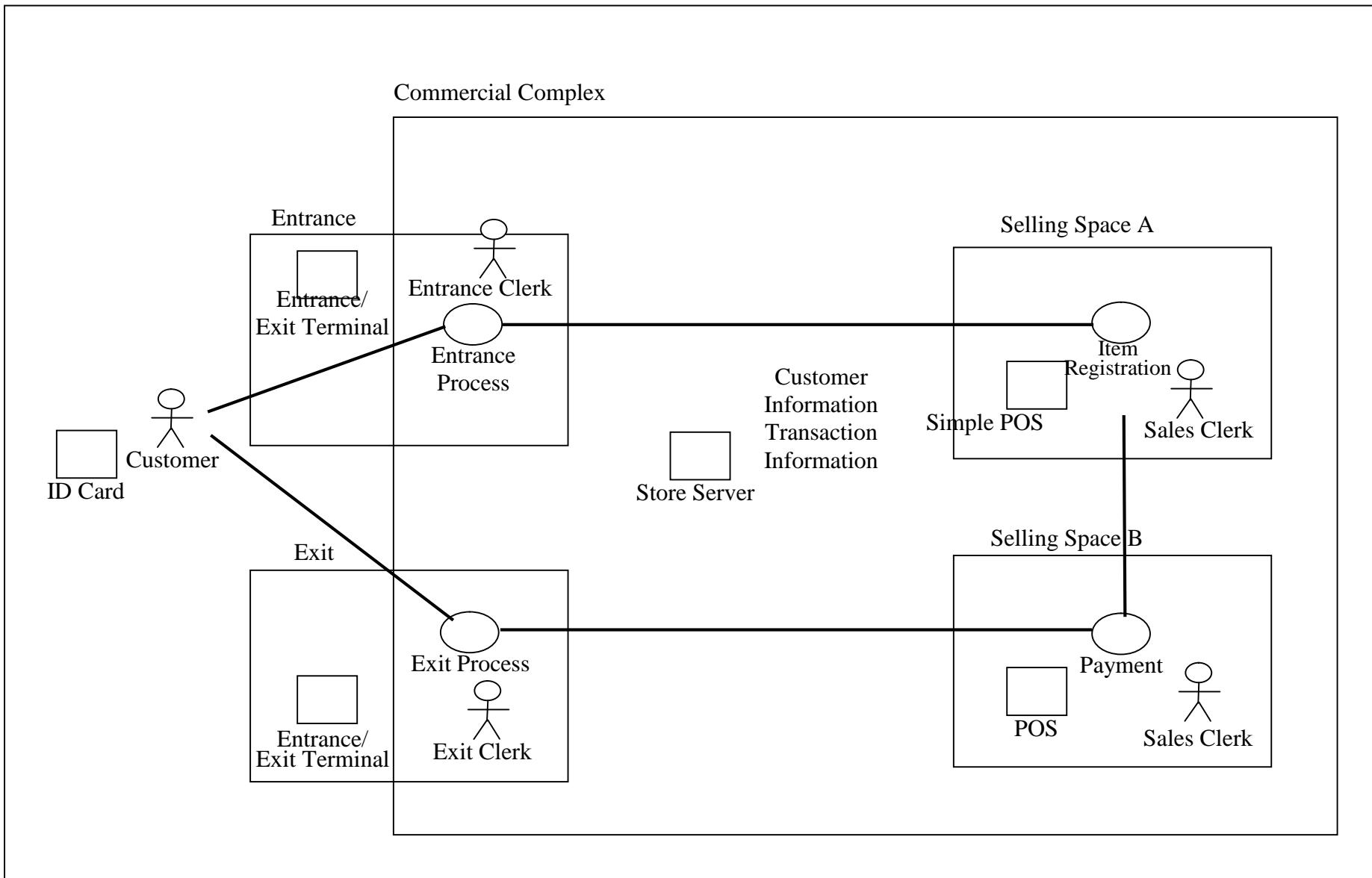
Definitions of entities:

Type	Entity	Description
Customer	Customer	
Clerk	Entrance Clerk	A clerk who performs an entrance process at an entrance of a commercial complex
	Exit Clerk	A clerk who performs an exit process at an exit of a commercial complex
	Sales clerk	A clerk who performs item registration process and payment process in a store in a commercial complex
Terminal	ID card	A card that identifies a customer Including member cards, credit cards, and instant issue cards

WS-POS 1.1 Technical Specification

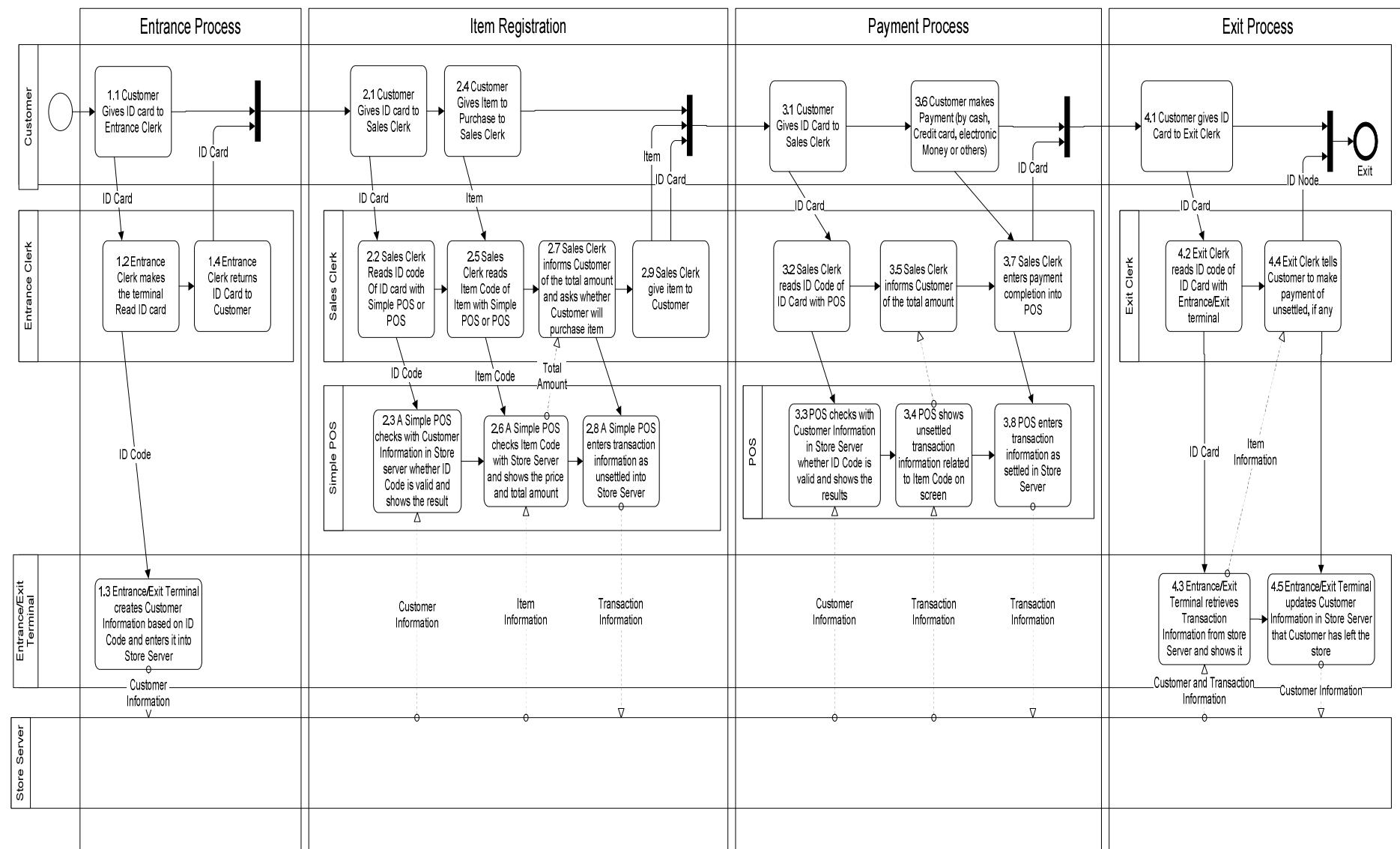
	Entrance/exit terminal	A terminal that reads or issues ID cards at an entrance or an exit A hand-held terminal equipped with a card reader for reading ID cards is assumed	
	Simple POS	A POS that can only perform an item registration process A hand-held terminal equipped with a card reader for reading ID cards and a bar code scanner for reading item bar codes is assumed.	
	POS	A POS that can perform an item registration process and a payment process A stationary POS equipped with a card reader for reading ID cards and a bar code scanner for reading item bar codes that can perform payment process is assumed	
	Store server	A server that centrally manages data of simple POS terminals and POS terminals in a store	
Data	Customer information	Information on ID cards, ID codes, and customers	
	Transaction information	Information that consists of ID codes, item codes and the payment status	

WS-POS 1.1 Technical Specification



WS-POS 1.1 Technical Specification

Activity Diagram



3.8 Sub Scope: Monitoring In-Store KIOSK Equipment and Cooperation with Back-office on Occurrence of Problems.

Contributor: Fujitsu Frontech Limited

The way of handling problems which might occur when a customer tries to let the machine read a card to make payment or to print out the result with POS printer after viewing various information (such as seat-reservation information for a movie theater or an airplane) on an in-store kiosk and reserving a ticket.

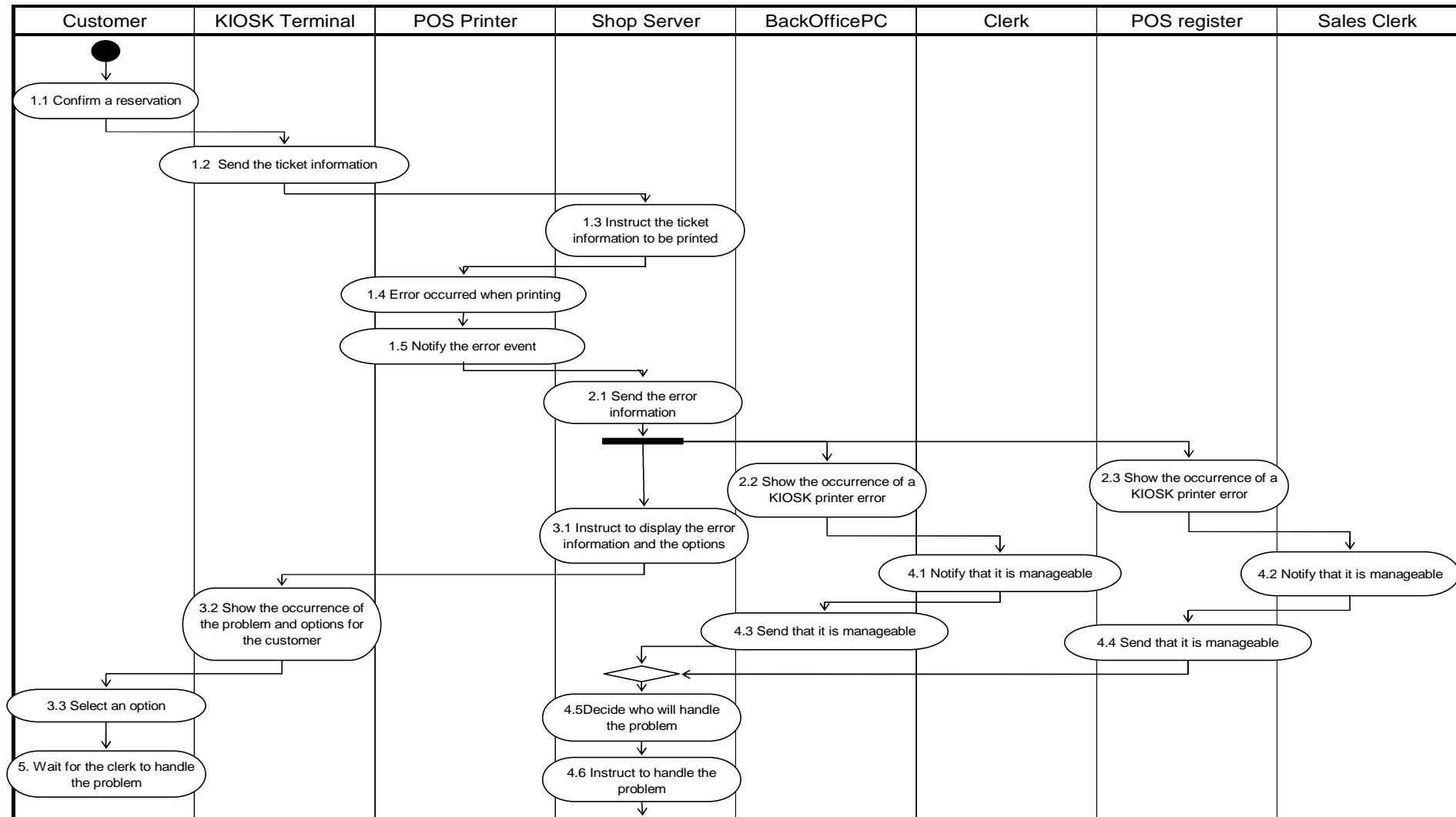
Definition of entities:

- | | |
|----------------------------------|---|
| • Customer | Customer is someone who reserves a ticket or purchases a product. |
| • Kiosk terminal 1 | Kiosk terminal is a terminal which controls the display of various kinds of information and the reservation of tickets. |
| • Kiosk terminal 2 | Multiple terminals are installed side-by-side and making up a department. |
| • Kiosk terminal 3 | |
| • POS printer | POS printer is a component of the kiosk terminal which outputs a receipt for a transaction or a receipt for ticket reservation. |
| • Ticket-reservation information | Ticket-reservation information is management information for seat-reservation information or others which are controlled by the kiosk terminal. |
| • Transaction information | Transaction information is information of actual results in accounting which is controlled by the POS register. |
| • Shop server | Shop server is a server which controls services on the kiosk terminal. |
| • Back-office PC | Back-office PC is a terminal which performs various tasks such as clerical works at the backyard of a store. |
| • POS register | POS register is a terminal used to make accounts for commodities. |
| • Sales clerk | Sales clerk is someone who operates the POS register. |
| • Clerk | Clerk is someone who performs various tasks except the operation of POS register. |
| • Electronic value R/W | Electronic value R/W is a component of the kiosk terminal which performs inputting and outputting of electronic cash and tickets. |

WS-POS 1.1 Technical Specification

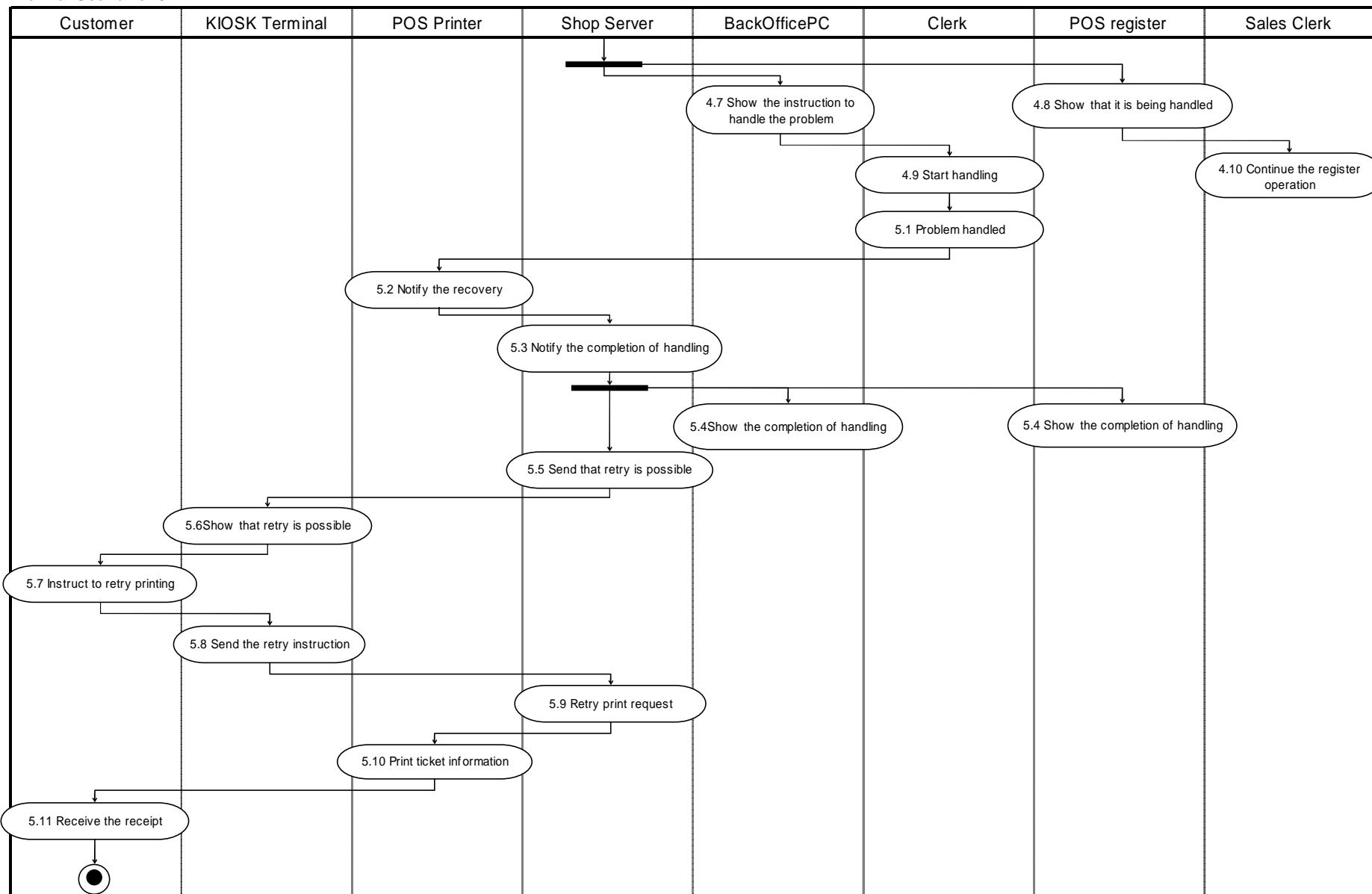
Activity Diagram

Normal Scenario 1/2



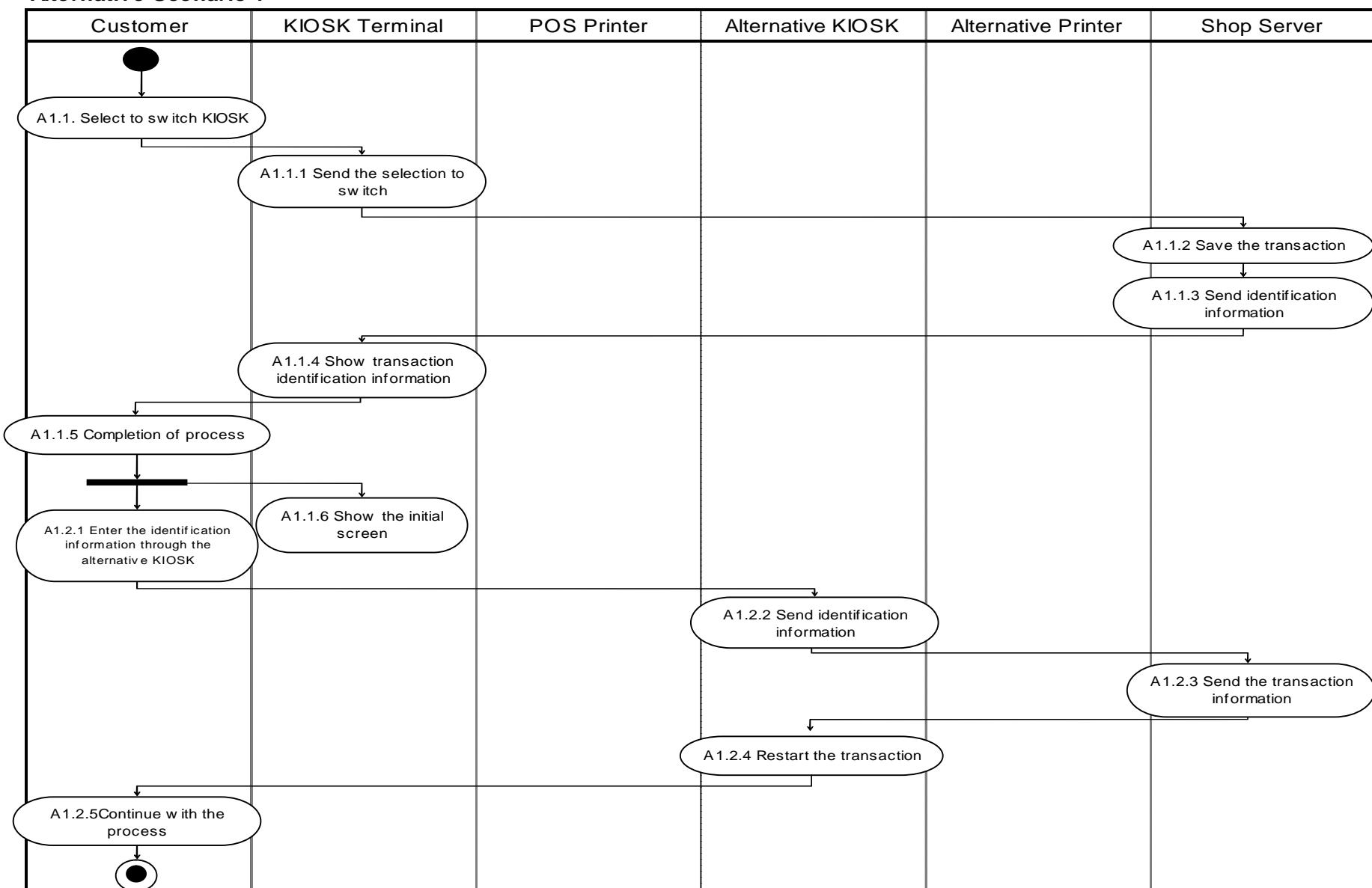
WS-POS 1.1 Technical Specification

Normal Scenario 2/2



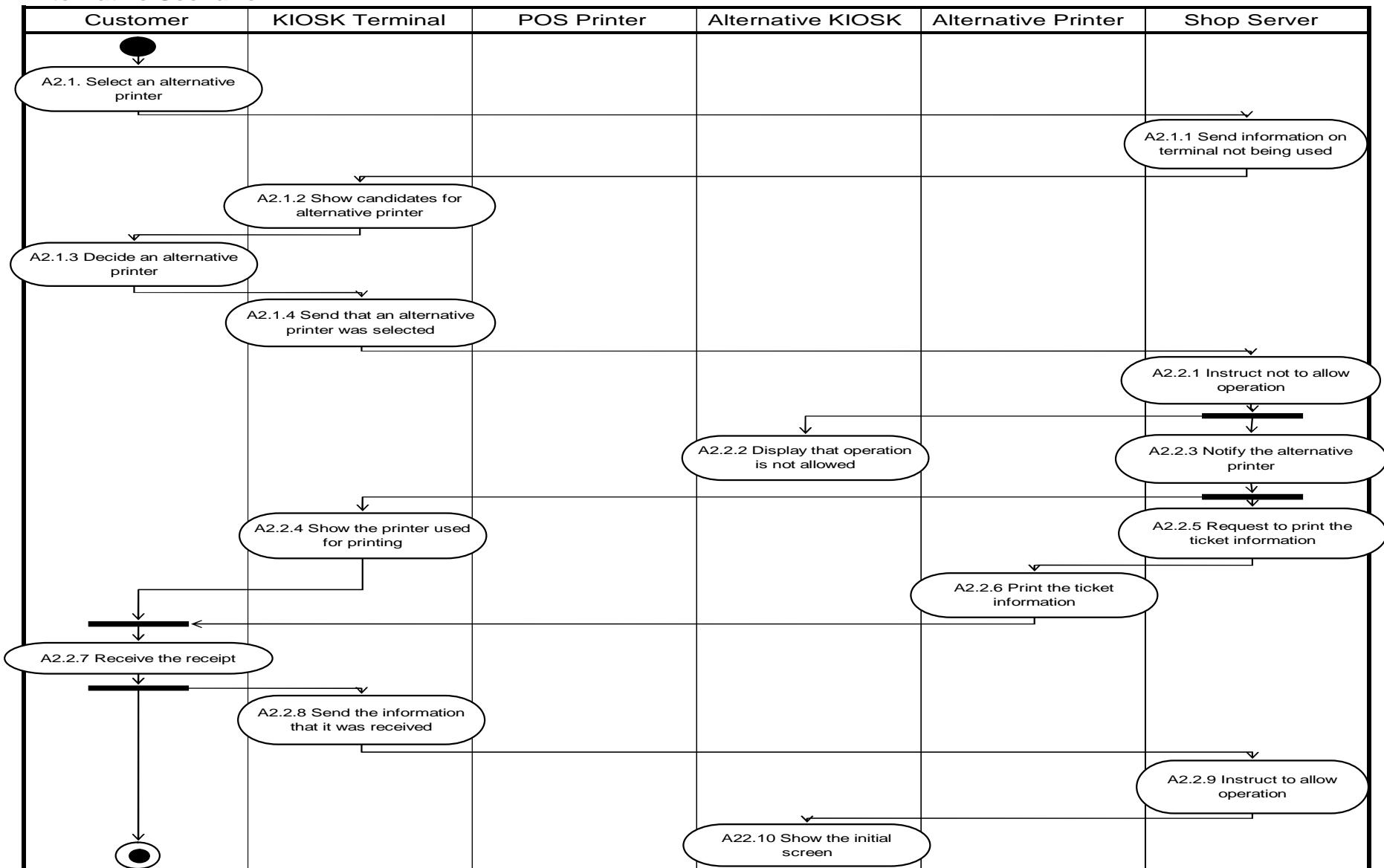
WS-POS 1.1 Technical Specification

Alternative Scenario 1



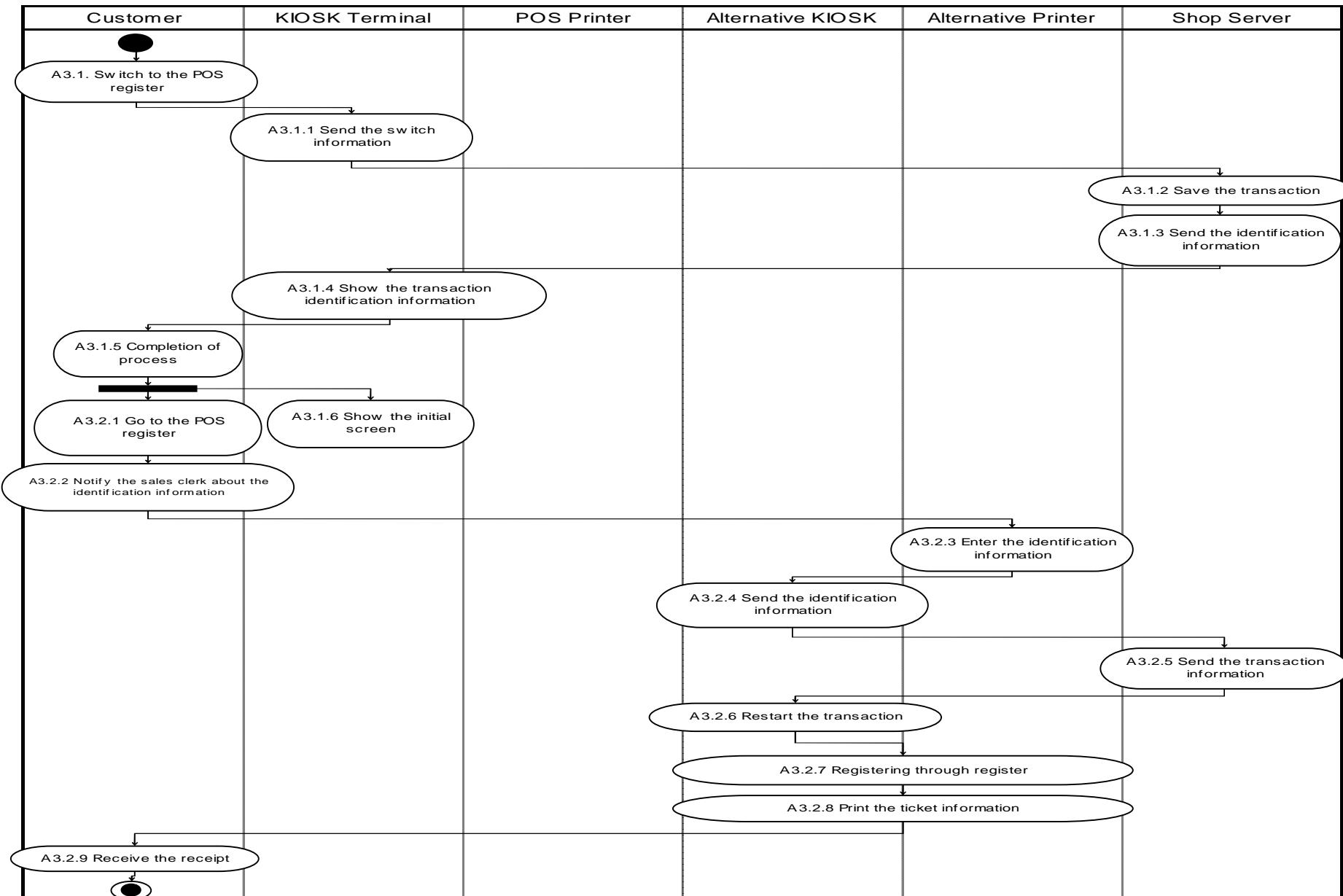
WS-POS 1.1 Technical Specification

Alternative Scenario 2



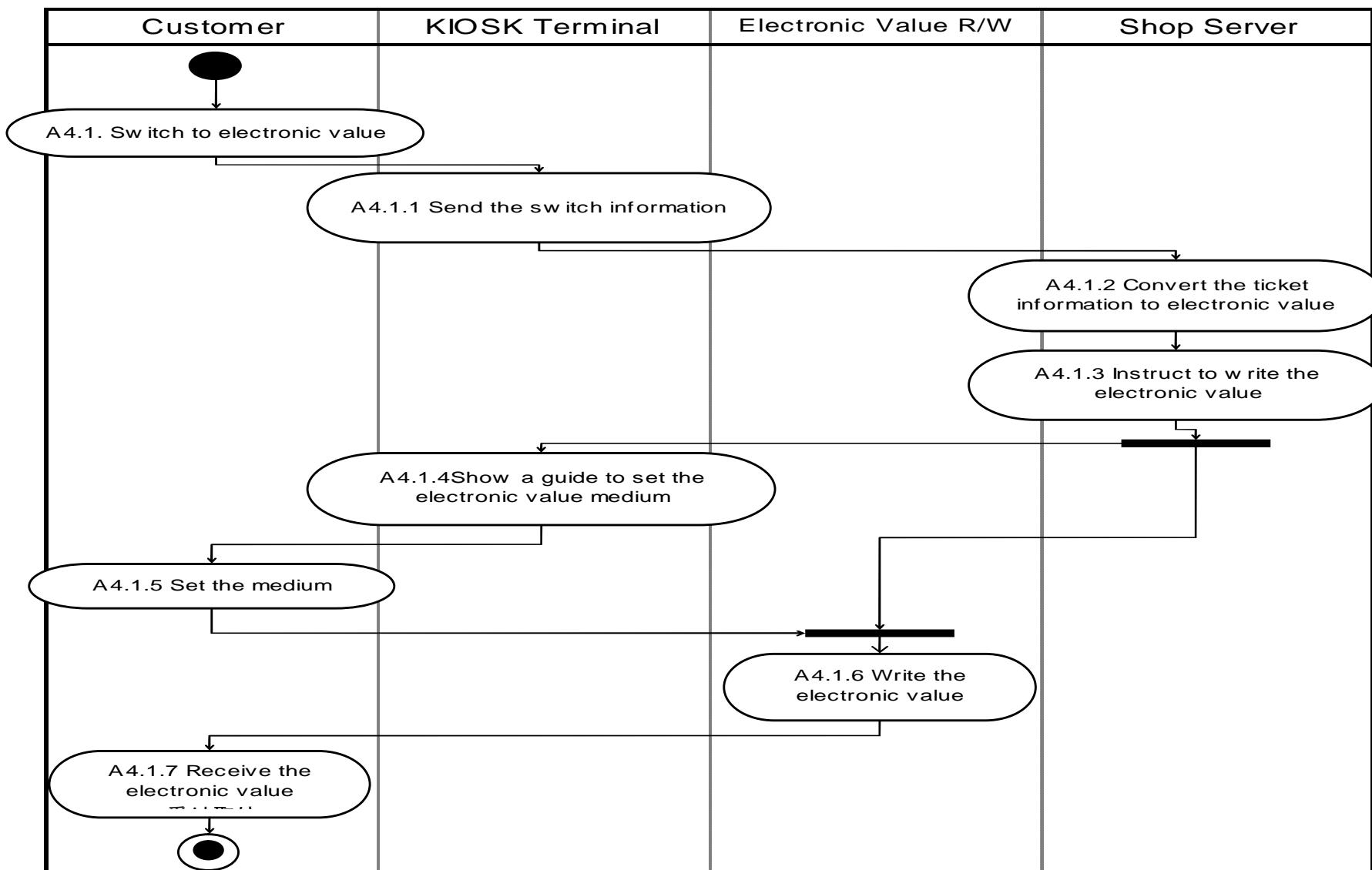
WS-POS 1.1 Technical Specification

Alternative Scenario 3



WS-POS 1.1 Technical Specification

Alternative Scenario 4



3.9 Sub Scope: POS System in Consideration of Cooperation between Various Industries

Contributor: Star Micronics Co., Ltd.

A customer purchases goods at a retail store or has a meal at a restaurant.

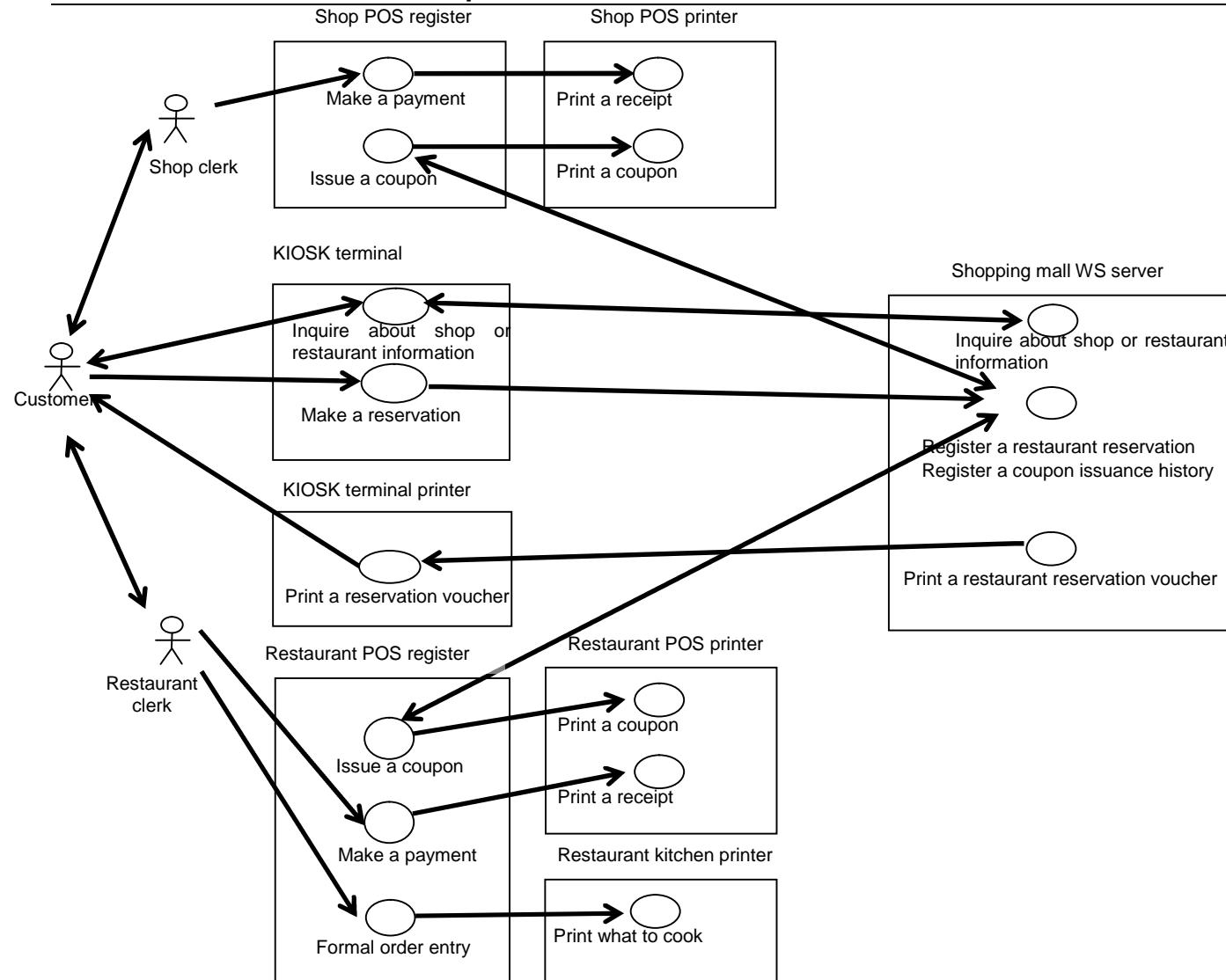
When making payment at a register, a coupon is issued which can be used at another industry (such as at a restaurant) along with issuing a receipt.

Reservation of seats at a restaurant or the like is possible with a kiosk installed within a shopping mall using this coupon.

Definition of entities:

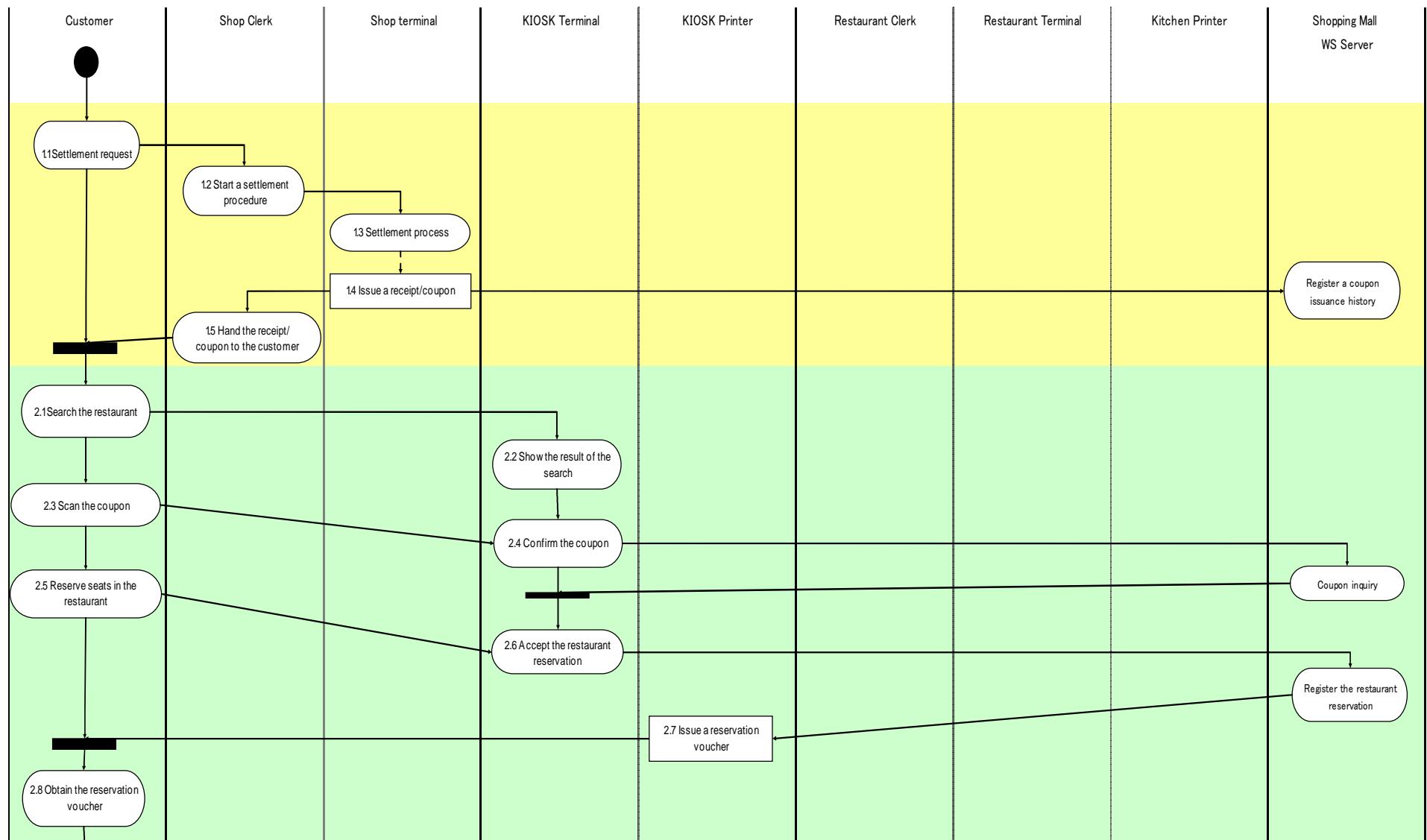
Customer	Someone who purchases goods or has a meal at a restaurant.
Shop clerk	Someone who operates the POS register in the shop.
Shop kiosk terminal	A terminal which retrieves and displays bargain information for shopping in the mall or which is used to reserve seats at a restaurant. This terminal is operated by the customer.
Shop information system	An information system which controls the receiving and placing of orders and performs customer management for the shop.
Shop POS terminal	A POS terminal installed in a shop. This terminal is operated by a clerk at the shop.
Restaurant clerk	Someone who arranges seats for customer and who receives order for dishes at a restaurant.
Restaurant information system	An information system which controls the receiving and placing of orders and performs seat management and other tasks in the restaurant.
Restaurant terminal	A terminal for the information system which controls the receiving and placing of orders and performs seat management and other tasks in the restaurant. This terminal is operated by a clerk at the restaurant.
Restaurant kitchen printer	A printer installed in the kitchen of a restaurant. It prints what to cook at the appropriate time.
Web Base Server System	A POS mutual mediation system shared within a shopping mall to support web-based data communication.

WS-POS 1.1 Technical Specification

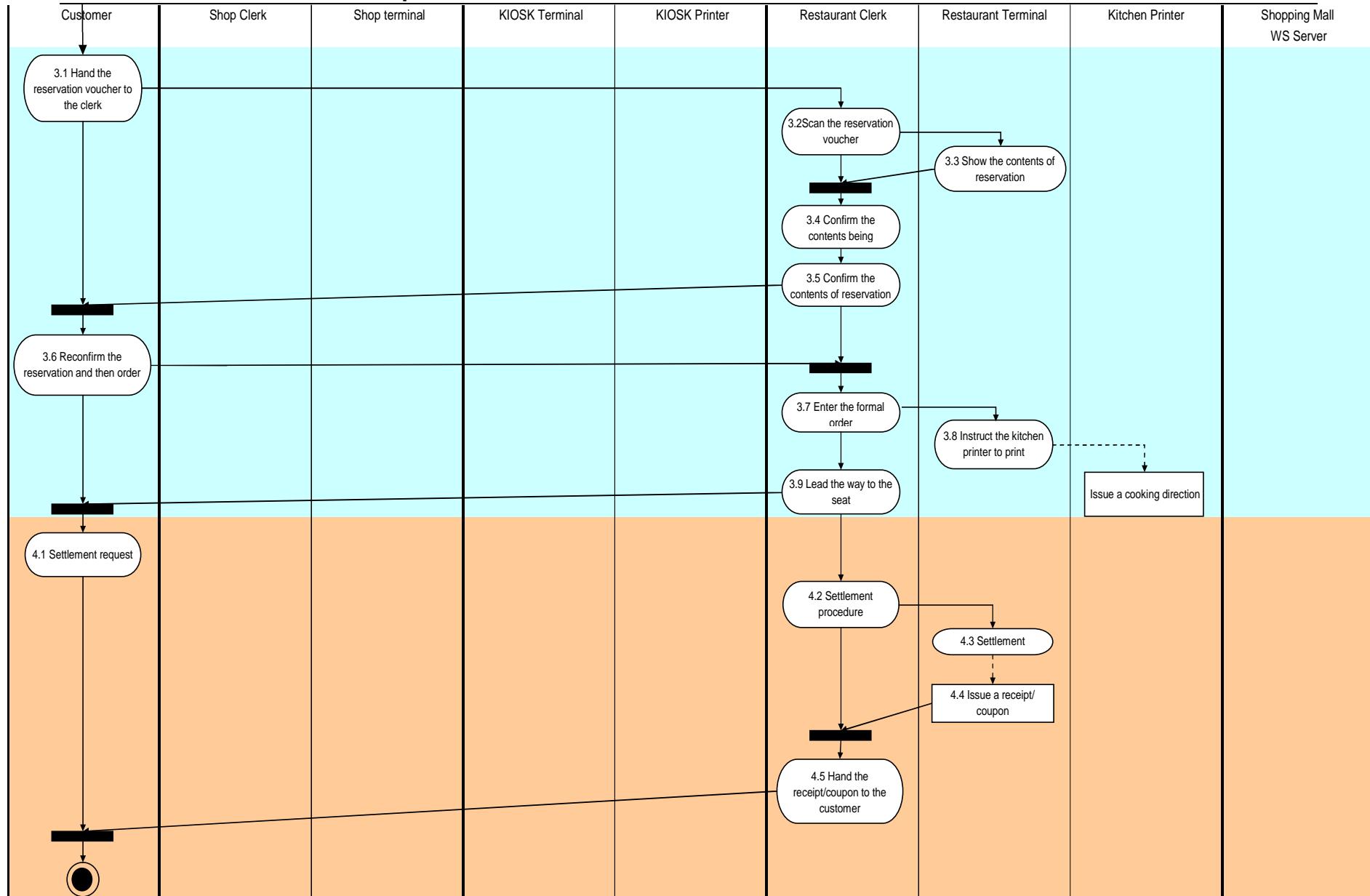


WS-POS 1.1 Technical Specification

Activity Diagram



WS-POS 1.1 Technical Specification



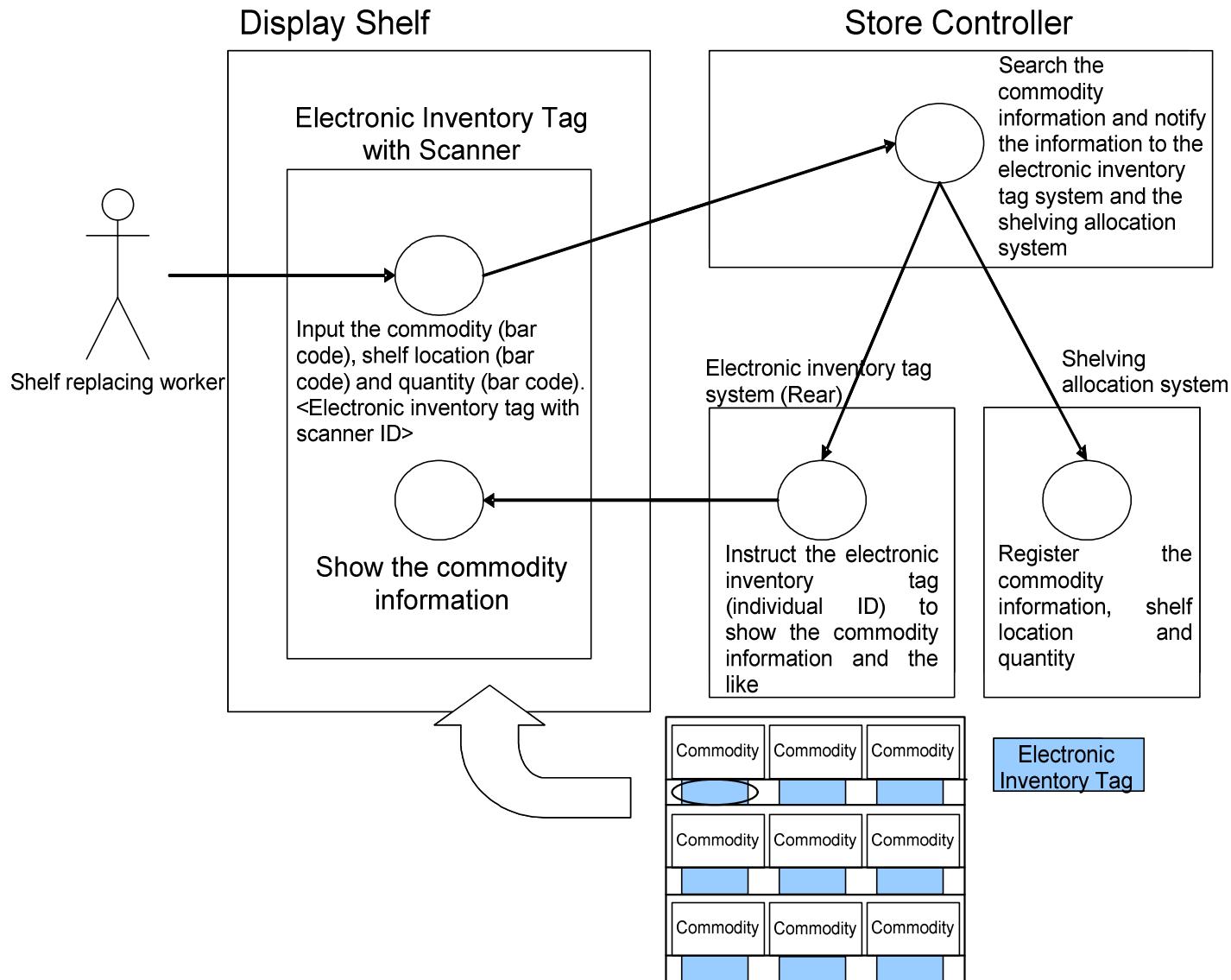
3.10 Sub Scope: Linkage between Display Shelf and Rear System

Contributor: OMRON SOFTWARE Co., Ltd.

Definition of entities (Part 1):

- Shelf replacing worker: A shelf replacing worker is someone who places relevant commodities on shelves while performing shelving allocation.
- Electronic inventory tag with scanner: An electronic inventory tag with scanner is a device which reads bar code indicating the product name and the location of the shelf and displays the commodity information such as commodity price (However, this device does not exist at present).
- Store controller: A server which controls commodity information (such as product name, selling price, quantity, and others).
- Shelving allocation system: A server which controls the state of shelving allocation on display shelf
- Electronic inventory tag system (Rear): A server which instructs the electronic inventory tag with scanner to display information and controls the state of display.

WS-POS 1.1 Technical Specification

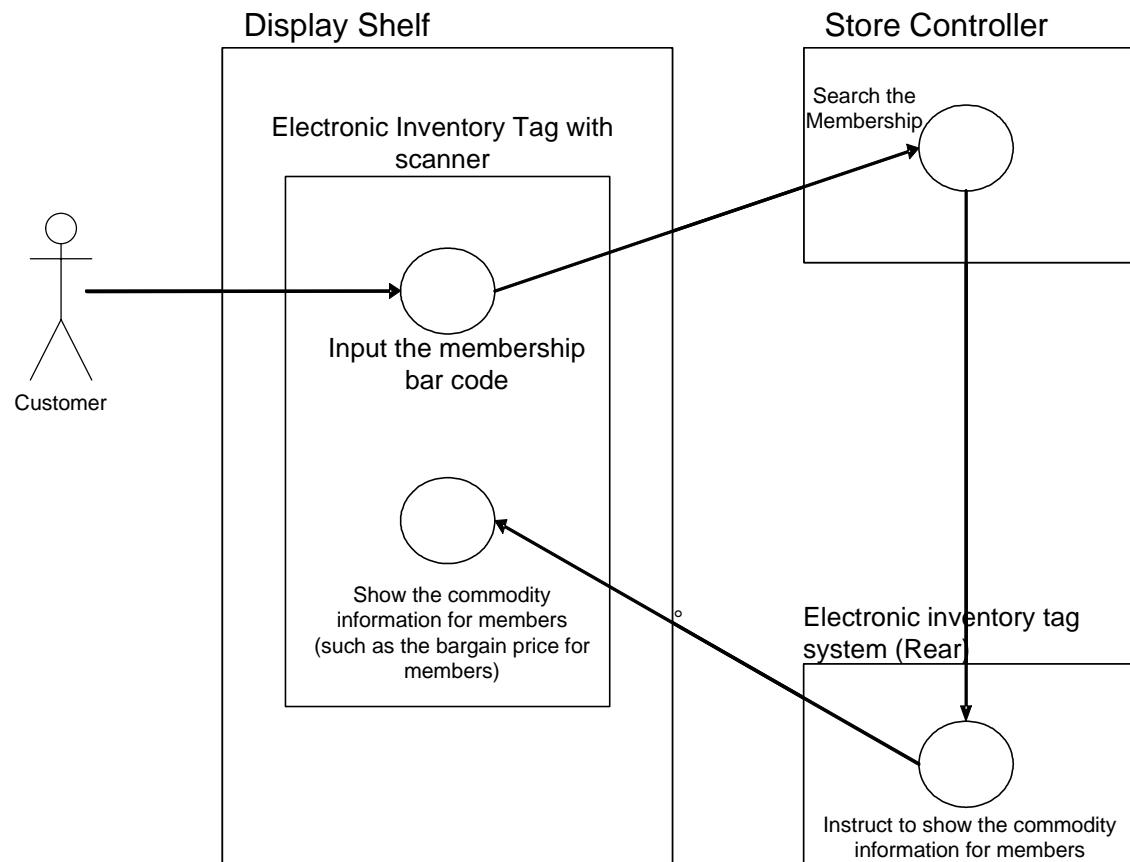


WS-POS 1.1 Technical Specification

Definition of entities (Part 2):

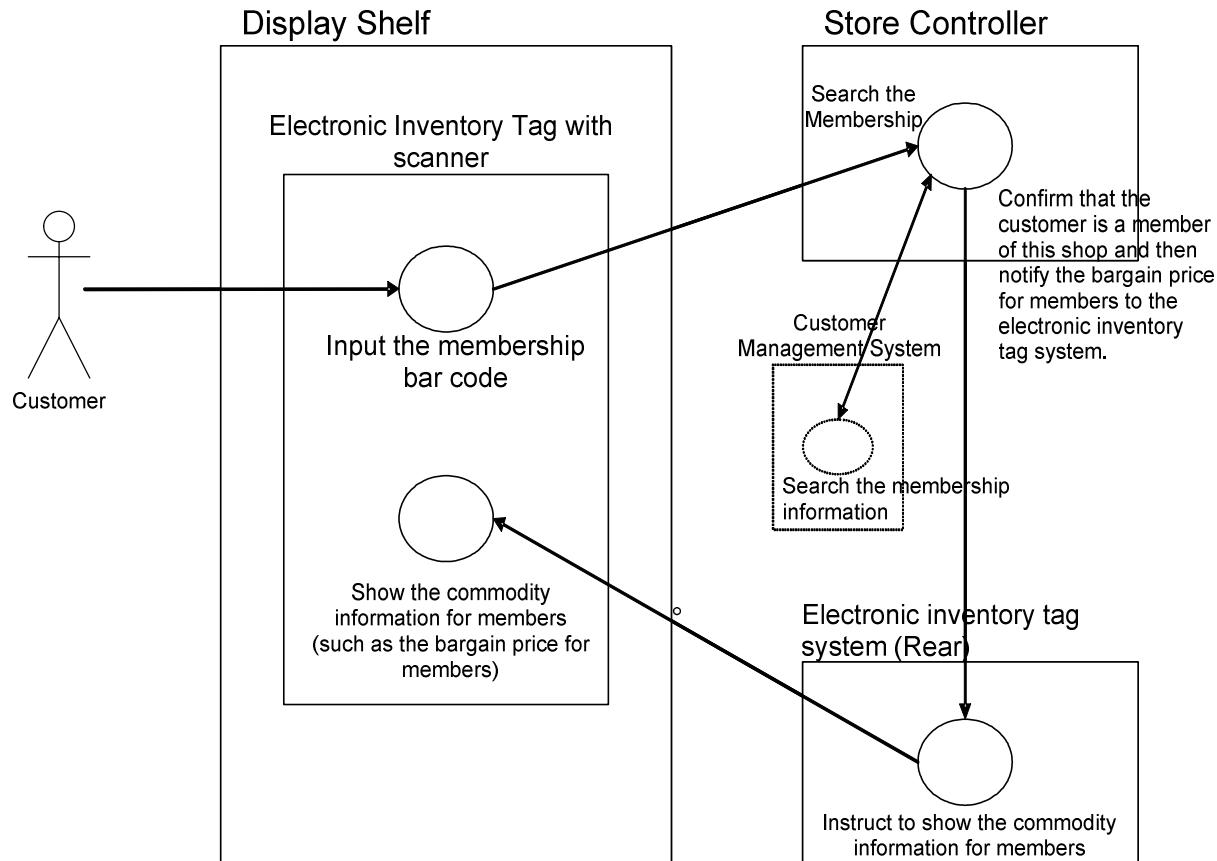
- Customer: A customer is someone who obtains information such as the selling price of the relevant commodity (including the selling price for members) from an electronic inventory tag.
- Electronic inventory tag with scanner: An electronic inventory tag with scanner is a device which reads the membership bar code and displays the commodity information such as the commodity price (However, this device does not exist at present).
- Store controller: A server which controls commodity information (such as product name, selling price, selling price for members, quantity, and others).
- Electronic inventory tag system (Rear): A server which instructs the electronic inventory tag with scanner to display information and controls the state of display.

WS-POS 1.1 Technical Specification



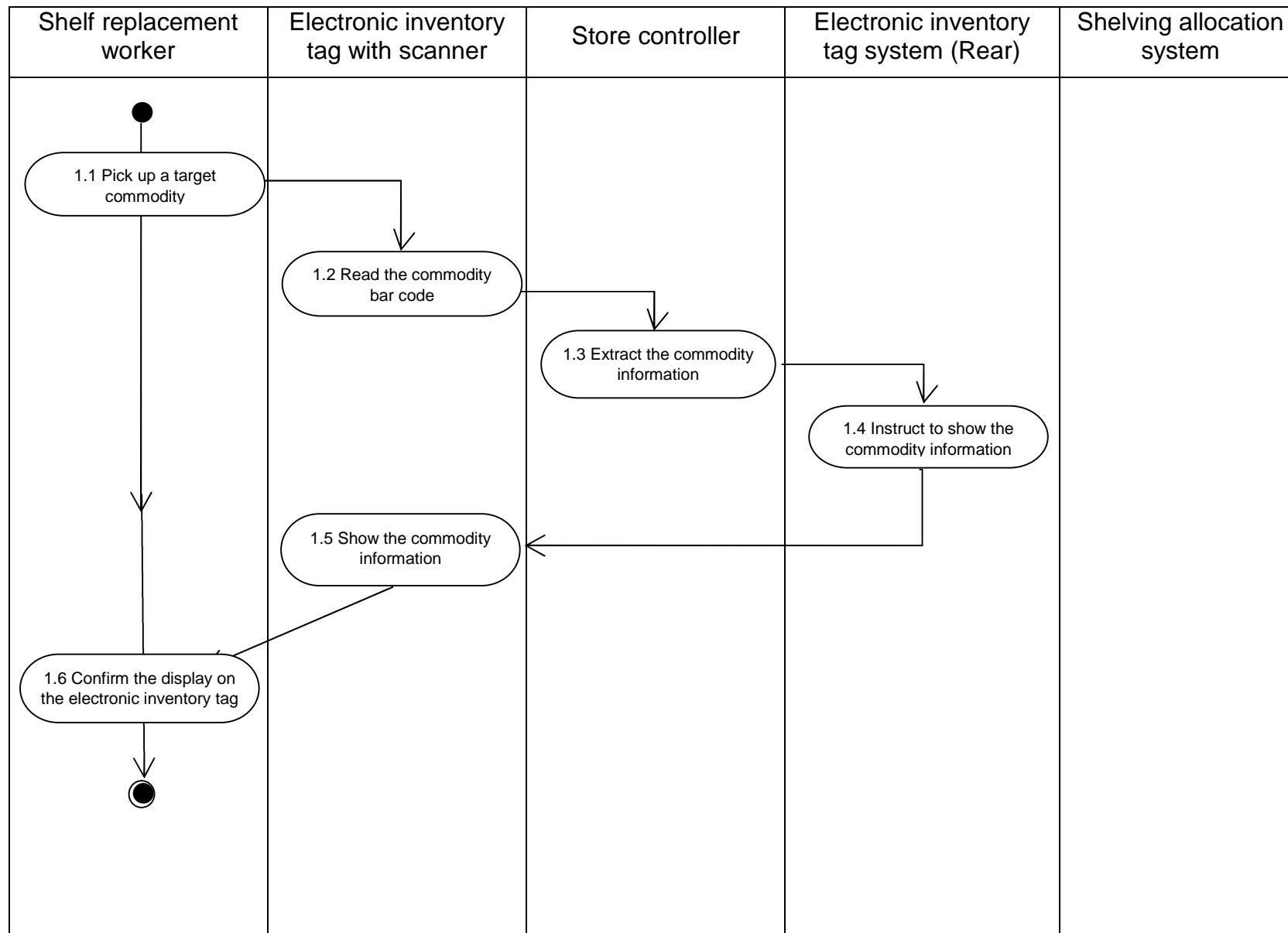
WS-POS 1.1 Technical Specification

<Alternative method>

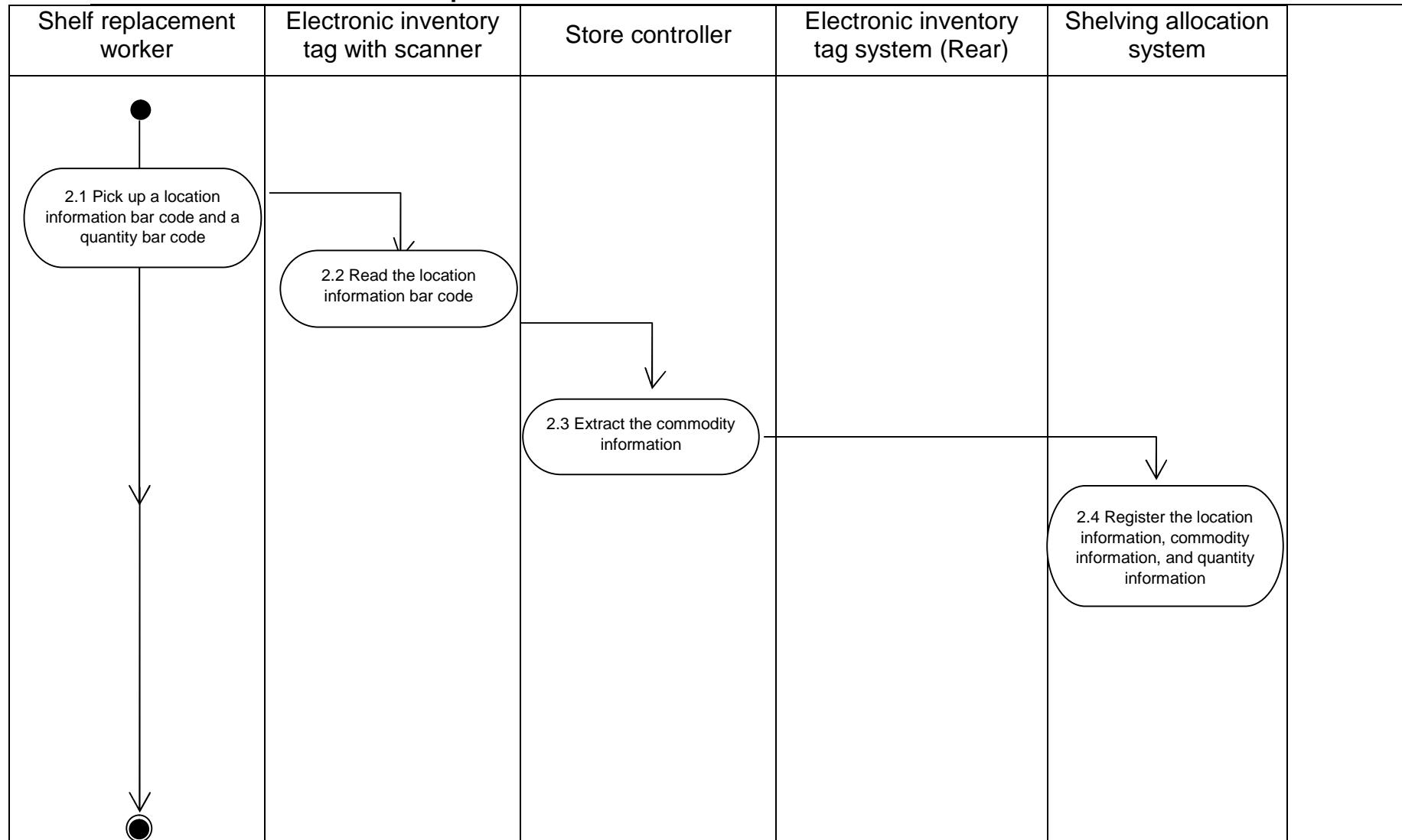


WS-POS 1.1 Technical Specification

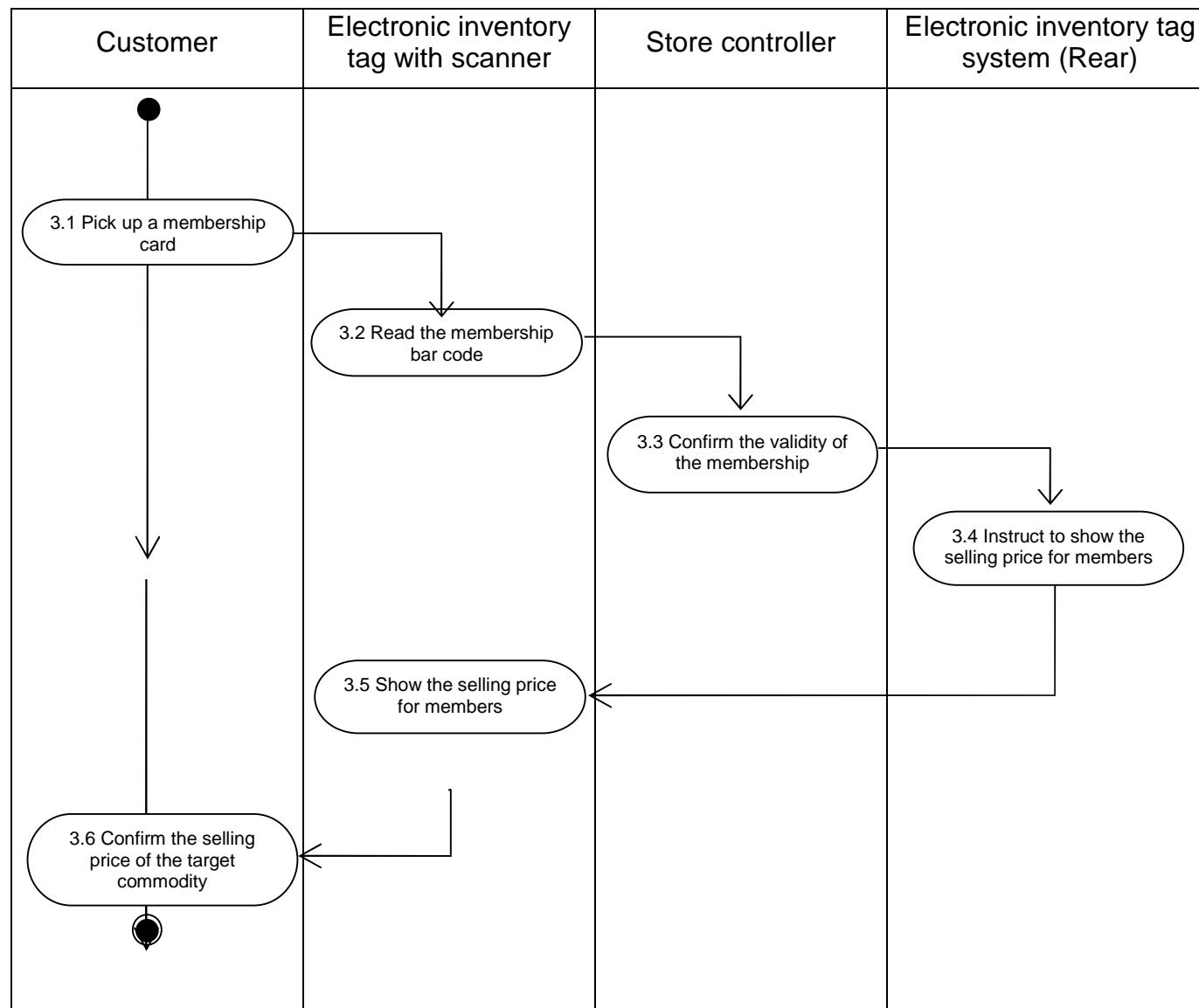
Activity Diagram



WS-POS 1.1 Technical Specification

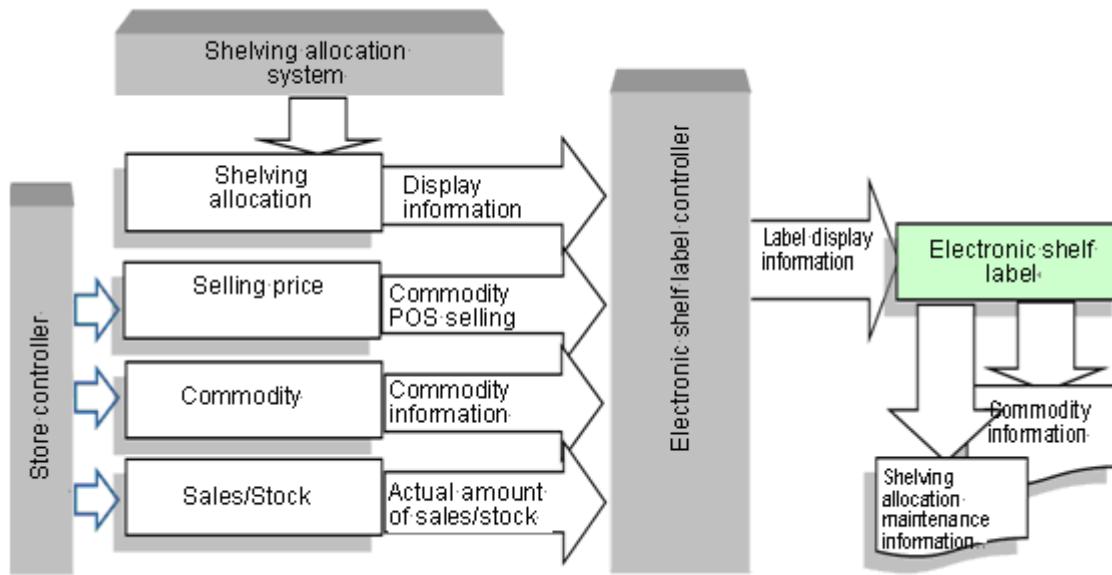


WS-POS 1.1 Technical Specification



3.11 Sub Scope: Cooperation between Electronic Shelf Label and Shelving Allocation Information

Contributor: Retail Science Co., ltd.



Target entities

- Shelving allocation (Shelving allocation system)
- Electronic shelf label controller
- Electronic shelf label device
- Shelving allocation maintenance information

Entities not targeted

- Store controller
- Selling price
- Commodity
- Sales/Stock

WS-POS 1.1 Technical Specification

Stakeholders and their concerns

Stakeholders	Concerns
Shelving Allocation Manager	A shelving allocation manager is in charge of creating and managing display data for each shop. A shelving allocation manager is interested in improving the productivity of a selling space by letting the shop carry out improvement or elimination of commodity correctly according to the plan.
Shop Operator	A shop operator is under an obligation to reflect the shelving allocation planned by the shelving allocation manager on actual shelves of the shop. He/she wishes to do this by carrying out the task efficiently and correctly to the extent possible.

Definition of Sub Scope

The group of entities related to the electronic shelf label shall be the overall scope. The management of the connection between the location information of an electronic shelf label and the commodity information, an interface between applications or between an application and a device used to display information on electronic shelf labels, and shelving allocation maintenance information displayed on electronic shelf labels shall be sub scope.

Definition of target entities

- Shelving allocation (Shelving allocation system)

An application which imports and maintains shelving allocation information created by an external shelving allocation system or the like for each selling space in each shop of a retailing company and provides a connection between an electronic shelf label device controlled by an electronic shelf label controller and location of a commodity displayed on a shelf of a shop. In some cases, the shelving allocation system is utilized by extending itself.

Conventionally, a main requirement for “shelving allocation” is to control commodity codes and location information in shelving allocation, but in this case, the shelving allocation supports electronic shelf label devices installed at a certain location on a shelf in providing commodity information by further controlling serial numbers of the electronic shelf label device, commodity codes, and location information of the commodity.

The “shelving allocation” has a trigger to reflect display information to the controller which controls the electronic shelf label device, or, the “shelving allocation” has a controller function in itself to provide information to the electronic shelf label device.

WS-POS 1.1 Technical Specification

- **Electronic shelf label controller**

This controller controls a serial number of an electronic shelf label device and information it displays. Electronic shelf label controller cooperates with “shelving allocation,” and sends information to be displayed on a specified electronic shelf label device to an electronic shelf label.

Representative items as information to be maintained are the basic information such as commodity name, price, and ordering unit, and the “shelving allocation maintenance information” used to support a shelving allocation maintenance work, which is a target in this case.

According to the performance of electronic shelf label device, in some cases, the electronic shelf label controller may have a control function which is in conformity with a means of communication dedicated to the device. And information such as commodity price is synchronized with master information or the current selling price by a POS system’s store controller via LAN.

- **Electronic shelf label device**

This is a device used to display information related to the commodity for customers or employees, and contents to be displayed are controlled by the electronic shelf label controller which manages an electronic shelf label’s serial number.

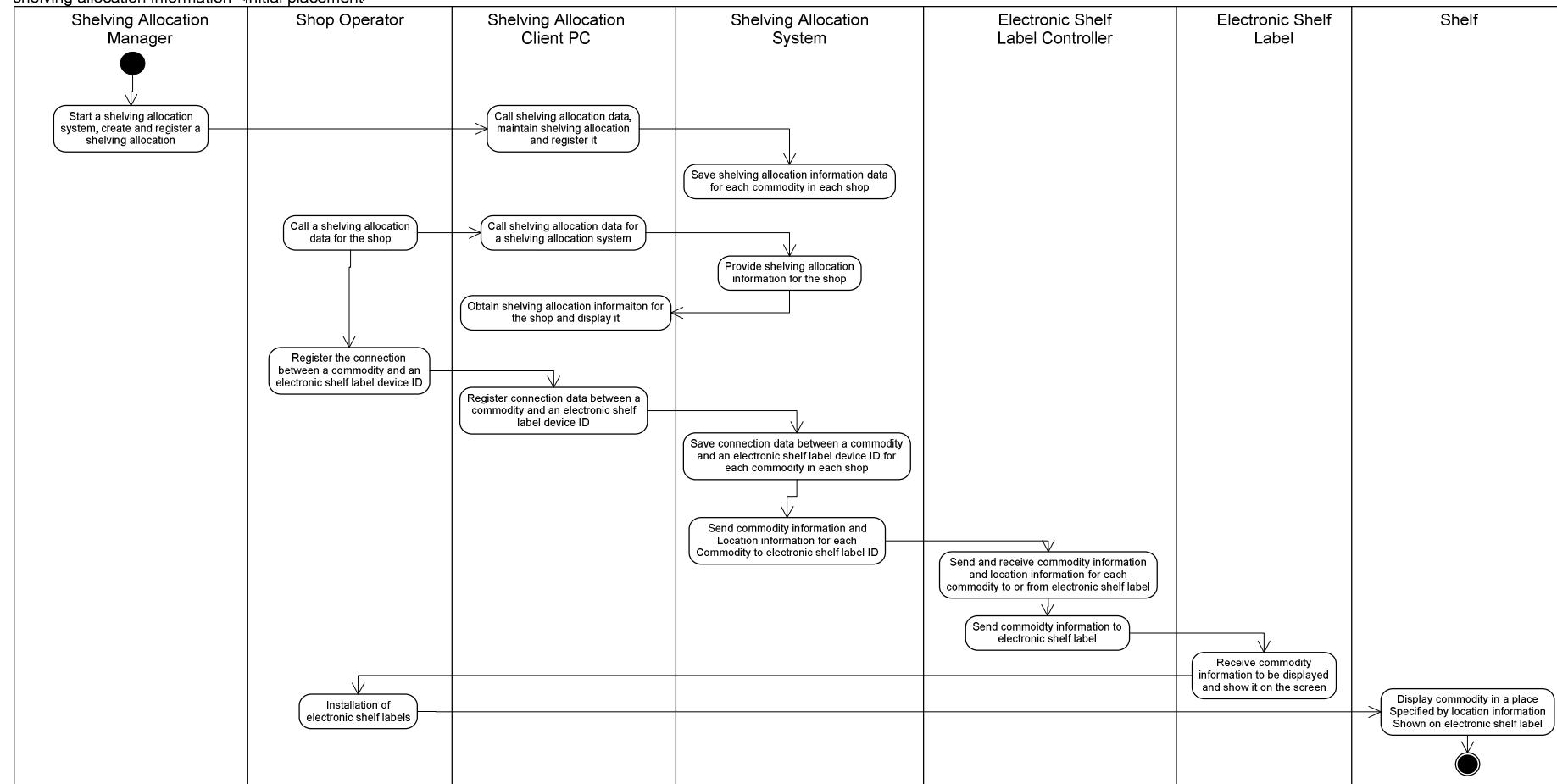
It is desirable to use a device which can display the commodity name and bar code on its LCD display or the like in consideration of an objective of this case. And it might be necessary to switch between information for customers and information for employees. As for the means of communication with an electronic shelf label device, it may be advantageous to be able to communicate directly with the device by using wireless LAN connection or Bluetooth connection in the future in consideration of its use for many purposes, but now, it is likely that the use of wired communication or wireless infrared communication via an electronic shelf label controller dedicated for the device will spread first in consideration of aspects such as the cost and popularity.

- **Shelving allocation maintenance information**

Shelving allocation maintenance information is information related to shelving allocation maintenance displayed on the electronic shelf label device. The commodity name, price, and commodity code (bar code) must be displayed on the electronic shelf label as an explanation of the commodity, but in addition to them, in this scope, information related to the shelving allocation maintenance, which is necessary for employees, (display location information (gondola number, shelf number, display order), number of faces, applicable period for the commodity) are also required.

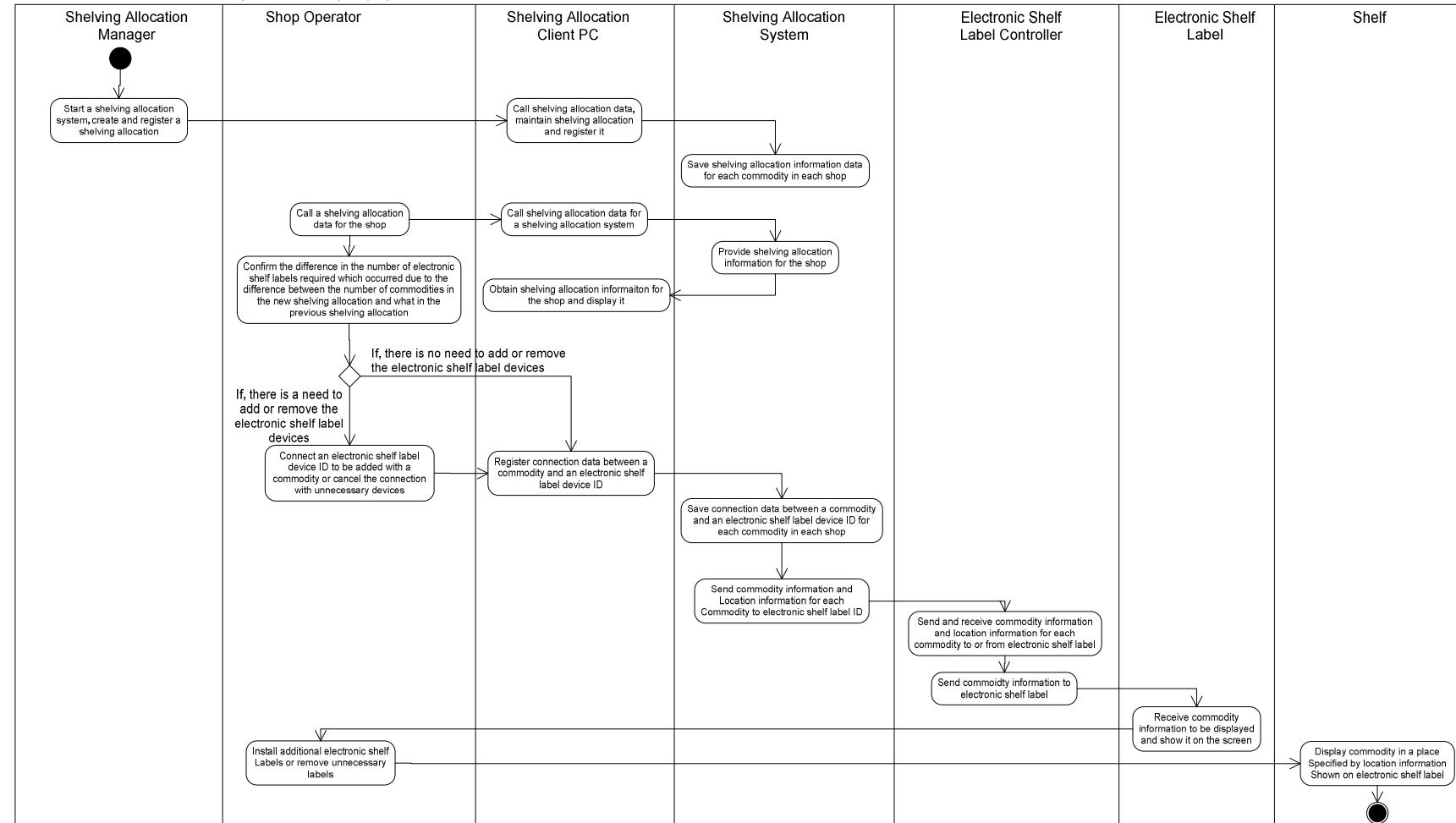
Activity Diagram (High level use case)

Cooperation between the electronic shelf label and the shelving allocation information <Initial placement>



Activity Diagram (Extended use case 1)

Cooperation between the electronic shelf label and the shelving Allocation information <Partial change in commodity display>



3.12 Sub Scope: Self-Service Refueling

Contributor: NEC Infrontia Corporation

Perform a self-service refueling transaction at a SS (Service Station).

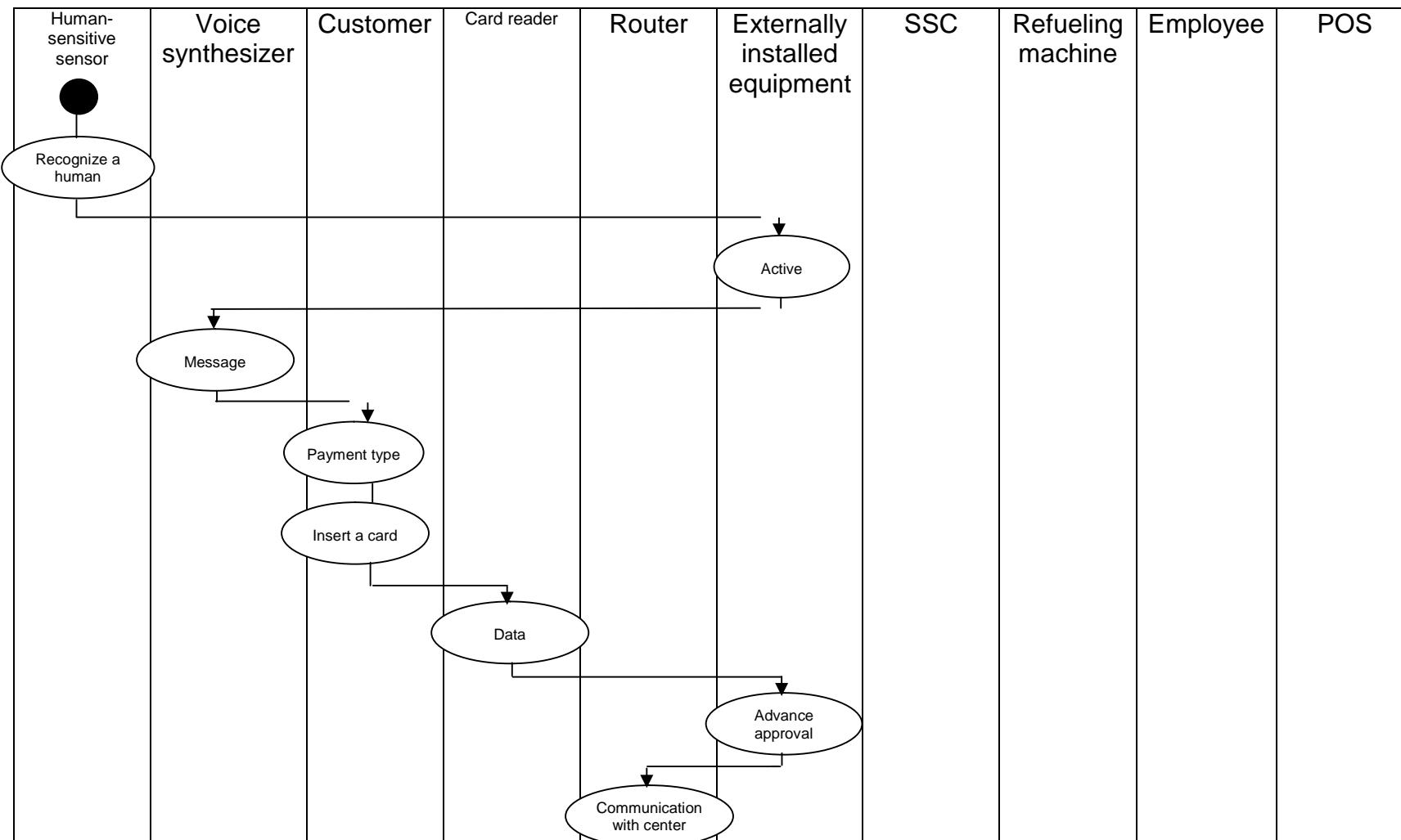
Definition of entities:

Customer	Performs a self-service refueling at a SS.
Car	Refueled.
Employee	In response to the refueling transaction by the customer, he/she allows refueling and stops refueling in a case of emergency through SSC.
Credit card	Performs a refueling transaction using a credit card.
Electronic money	Performs a refueling transaction using electronic money.
Cash transaction	Performs a refueling transaction using cash.
SSC (Self Service Console)	Controls refueling equipment by allowing or prohibiting a refueling.
POS	Performs a refueling transaction, ordering, stocking, single item sales, card issuance, customer management, and commodity management.
Card reader	Reads and issues magnetic cards.
Contactless reader	Handles electronic money.
Human-sensitive sensor	Activates externally installed equipment when someone approaches it.
Voice synthesizer	Lets the customer know what to do next with voice.
Printer	Prints a receipt when a refueling is completed.
Payment machine	Puts money in it for settlement in the case of cash transaction.
Change machine	Dispenses the change when a refueling transaction is completed.

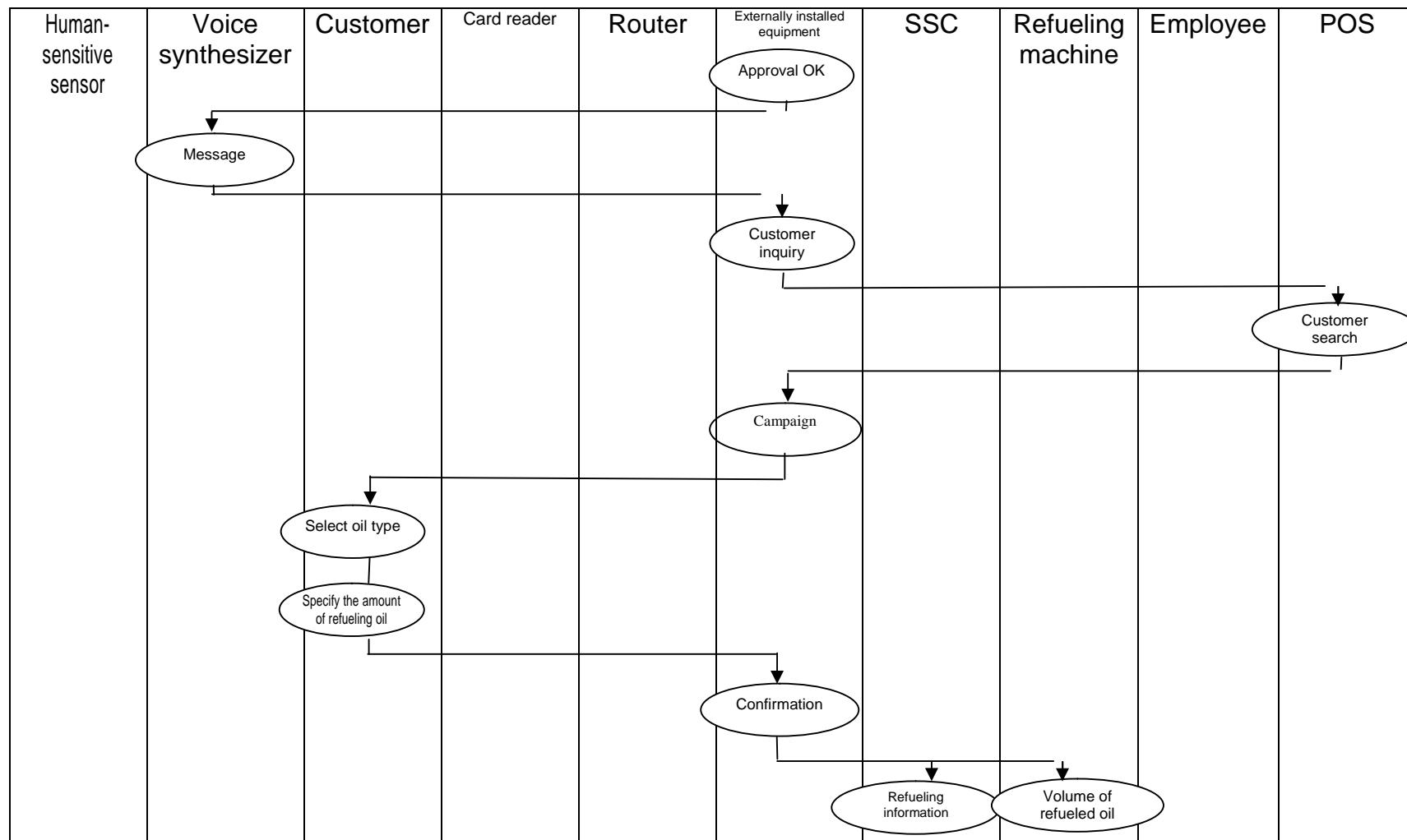
WS-POS 1.1 Technical Specification

Externally installed equipment	Specifies with it the type of settlement, type of oil, and amount of oil outdoors. Performs an advance approval process in the case of credit card settlement. Performs a credit confirmation in the case of electronic settlement. Performs a money-receiving process and instructs the change machine to dispense the change in the case of cash settlement. Outputs a receipt through the printer when a refueling is completed.
Measuring machine	Measures the amount of oil refueled.
Refueling machine	Refuels cars.
Oil level indicator	Measures the remaining amount of each type of oil in tank.
Car-washing machine	Washes cars.
Router	Performs communication with a credit center and a totalization center.
Forecourt Device Communication BOX	Acts as a bridge between POS, externally installed equipment, SSC, and Forecourt devices.
Shop server	Performs customer management, outputting of sales management report, and exchanging information with the primary distributor.
Customer information	Information related to a car (Maintenance, vehicle inspection, oil change) Management of information related to a customer (Hobby, address, family makeup, age, etc.)
Transaction data	Amount of refueled oil, payment type, unit price.

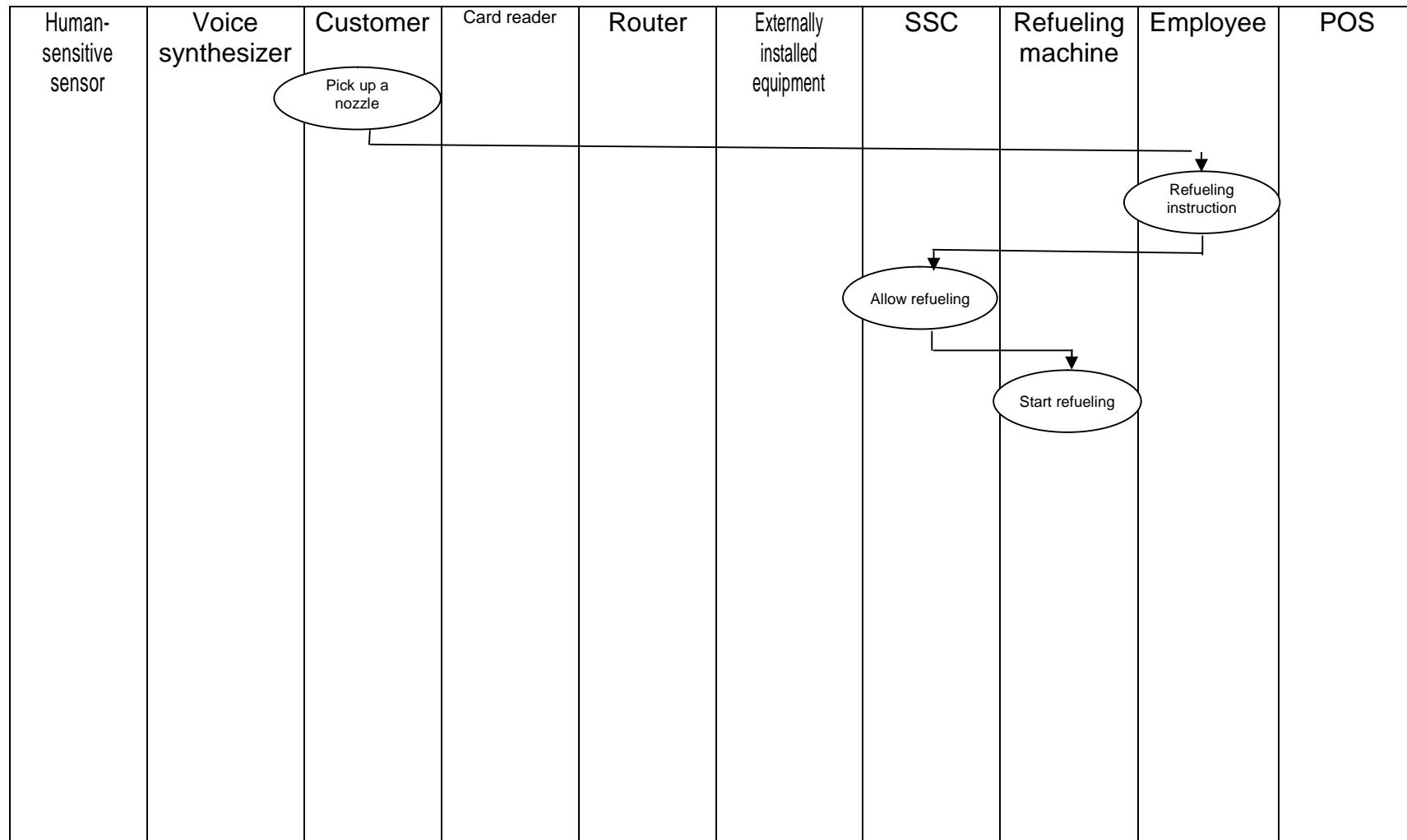
Activity Diagram



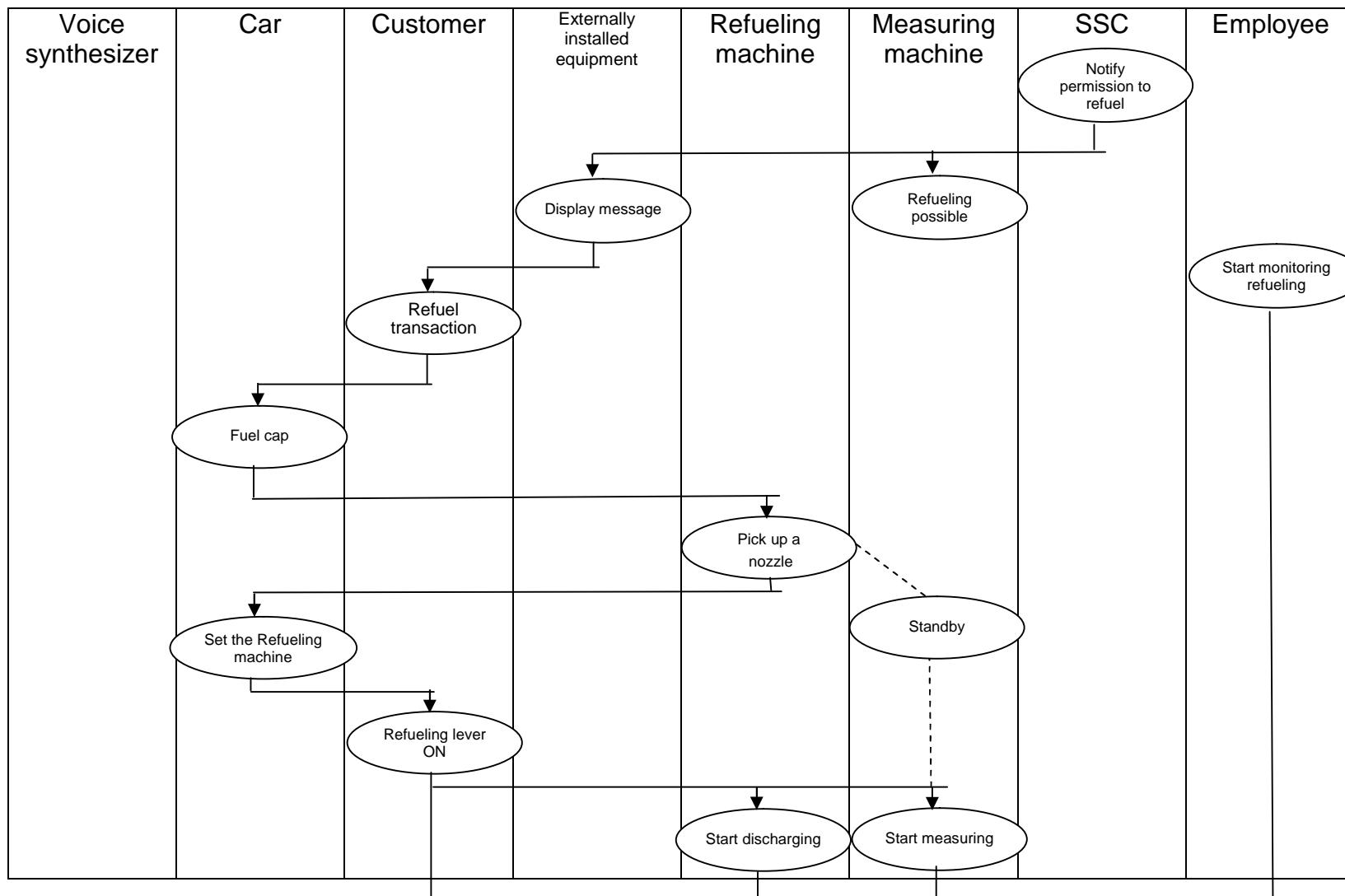
WS-POS 1.1 Technical Specification



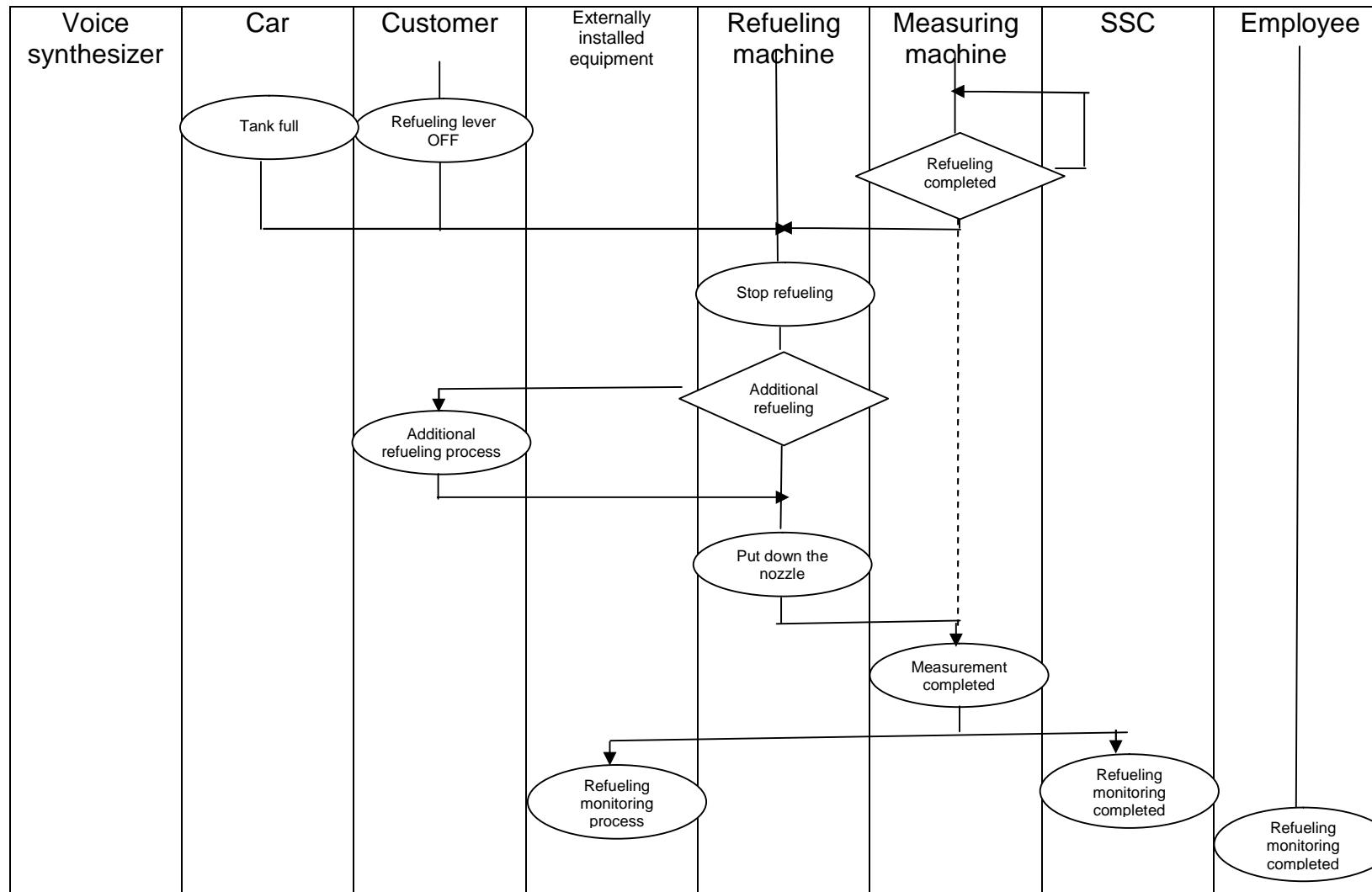
WS-POS 1.1 Technical Specification



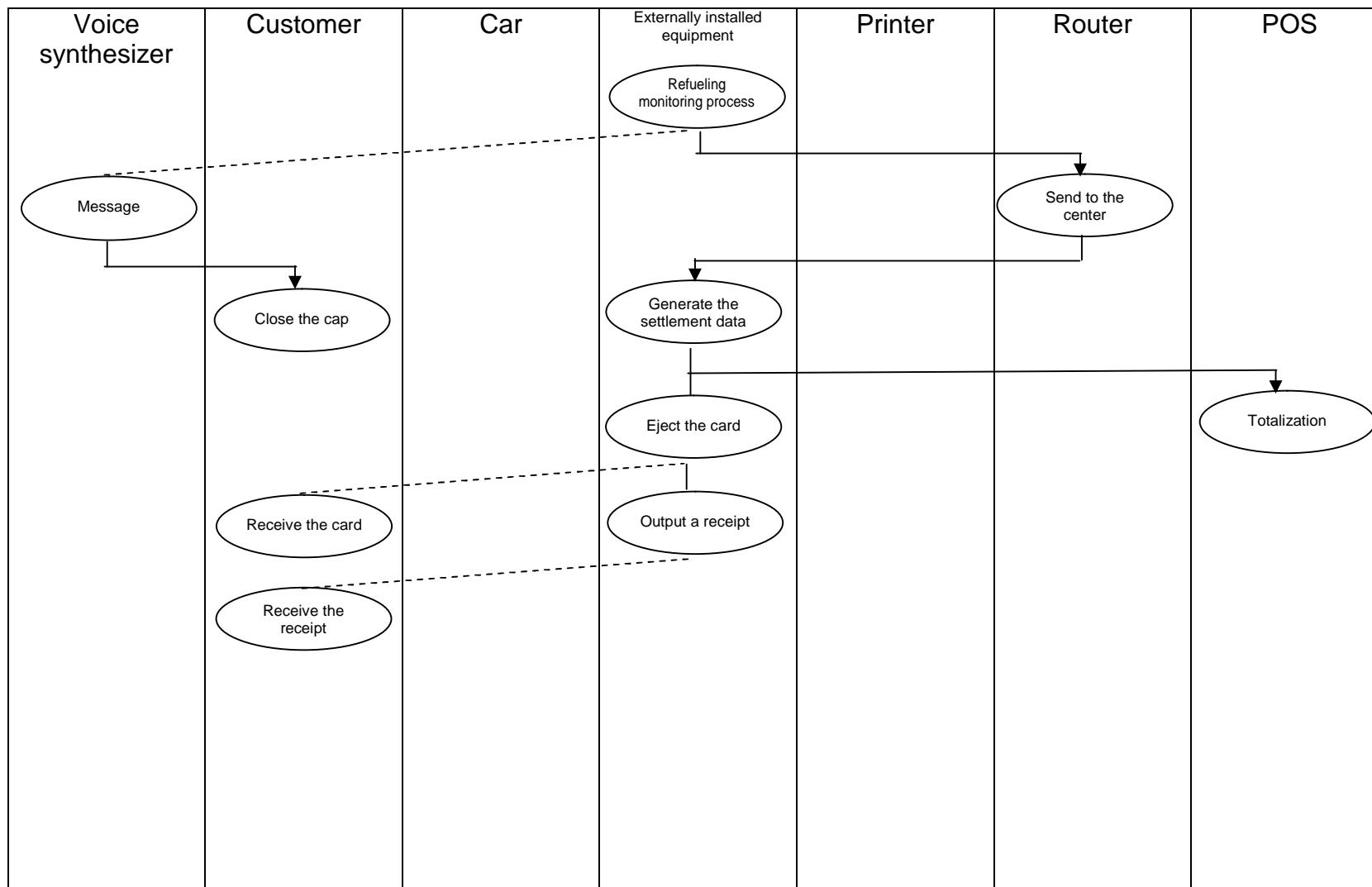
Activity Diagram



WS-POS 1.1 Technical Specification



Activity Diagram



4. DOCUMENT HISTORY

Version History

Ver	Date	Sections	Description of Change
0.1	2007-09-02	All	Initial document creation
0.2	2007-09-11	All	Fixed references, reworded, split components between WS and other
0.3	2008.07.31	All	Reorganized doc, added notes about what sections should contain
0.4	2008.08.26	Security, Flow	Added specific security requirements, WS-POS Flow section, Out-of-scope
0.5	2008.09.24	All	Events, Security, Service Description, Discovery
1.1	2011.05.15	All	Updated document to reflect inputs from field beta testing provided by OPOS-J; updated support for WSDL generation using WCF and JAX-WS and revised XMLPOS to ensure interoperability; updated documentation for clarity; added WSPOS class diagram .Net generated examples for all UnifiedPOS 1.13.1 peripheral devices. Added software support files to aide developers.

5. GLOSSARY

Term	Definition

6. REFERENCED DOCUMENTS AND SOFTWARE SUPPORT FILES

6.1 Referenced Documents

Term	Definition
WS-I Basic Profile V1.2	http://www.ws-i.org/Profiles/BasicProfile-1.2.html
SOAP 1.1	http://www.w3.org/TR/2000/NOTE-SOAP-20000508/
SOAP HTTP Binding	http://www.w3.org/TR/2007/REC-soap12-part2-20070427/#soapinhttp
SOAP HTTP Status Codes, Error Conditions	http://www.w3.org/TR/2007/REC-soap12-part2-20070427/#http-reqbindwaitstate
WS-Addressing 1.0	http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/
WSDL 1.1	http://www.w3.org/TR/wsdl
WS-MetadataExchange Publication	http://specs.xmlsoap.org/ws/2004/09/mex/WS-MetadataExchange.pdf
UDDI V3	http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm
TLS V1.2	http://www.ietf.org/internet-drafts/draft-ietf-tls-rfc4346-bis-04.txt
WS-Security V1.1	http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf
Using WSDL in a UDDI Registry, Version	http://www.oasis-open.org/committees/uddi-

2.0.2	spec/doc/tn/uddi-spec-tc-tn-wsdl-v2.htm
UnifiedPOS 1.13	www.NRF-ARTS.org

6.2 Software Support Files

The following software SDK files are available (See: www.NRF-ARTS.org) to aid Developers in the creation of WS-POS 1.1 compliant applications.

WS-POS Ver. 1.1 File Lists

- 1) WS-POS 1.1 equivalent XML Schema Files (XMLPOS 1.13.1 is used)

File Name: WS-POS1.1XSD.zip

- 2) WS-POS 1.1 equivalent WSDL Files (Independent files and implementation dependent files)

File Name: WS-POS1.1WSDL.zip

- 3) WS-POS 1.1 WCF Contract Files

File Name: WS-POS1.1WCFContract.zip

- 4) WS-POS 1.1 JAX-WS Contract Files

File Name: WS-POS1.1JAX-WSContract.zip

- 5) WS-POS 1.1 WCF Sample Program

File Name: WS-POS1.1WCFSample.zip

- 6) WS-POS 1.1 JAX-WS Sample Program

File Name: WS-POS1.1JAX-WSSample.zip

Important Notice For Condition of Use:

ARTS and User agree that the Software is provided "AS IS" and that ARTS makes no warranty as to the Software. ARTS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, RELATED TO THE SOFTWARE, ITS USE OR ANY INABILITY TO USE IT, THE RESULTS OF ITS USE AND THIS AGREEMENT.

7. WS-POS CLASS DIAGRAMS

The following are examples of the Class Diagrams that describe the WS-POS services that support the thirty-six POS peripheral devices that are described in the UnifiedPOS specification. They are included here to help in the understanding of the interrelationship of the processes and data that is necessary for a web services implementation.

Important Note Concerning the Following UML Examples

The included class diagram examples were distilled from C# code using the Microsoft Visual Studio 2010 development tools. As a result, the programmatic dependent language features are valid for C# only. A similar set of Class Diagrams for a Java implementation could also be generated using an equivalent Java development tool but the programmatic dependent language features would be different from what is shown.

The intent here is to show the relative relationship of the UnifiedPOS defined Properties, Methods, Events, and Data Parameters using standard UML imagery. The UnifiedPOS documentation should be consulted for detailed Peripheral Device operational information.

The peripheral devices are shown in alphabetical order and equate to the respective peripheral device chapters in the UnifiedPOS standard.

Belt

Belt Interface	
Methods	
	AdjustItemCount(BeltDirection Direction, int Count) : void
	CheckHealth(HealthCheckLevel Level) : void
	Claim(int Timeout) : void
	Close(string EndpointAddress) : void
	CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
	DirectIO(int Command, int Data, object Obj) : DirectIOData
	GetAutoStopBackward() : bool
	GetAutoStopBackwardDelayTime() : int
	GetAutoStopBackwardItemCount() : int
	GetAutoStopForward() : bool
	GetAutoStopForwardDelayTime() : int
	GetAutoStopForwardItemCount() : int
	GetCapAutoStopBackward() : bool
	GetCapAutoStopBackwardItemCount() : bool
	GetCapAutoStopForward() : bool
	GetCapAutoStopForwardItemCount() : bool
	GetCapCompareFirmwareVersion() : bool
	GetCapLightBarrierBackward() : bool
	GetCapLightBarrierForward() : bool
	GetCapMoveBackward() : bool
	GetCapPowerReporting() : PowerReporting
	GetCapSecurityFlapBackward() : bool
	GetCapSecurityFlapForward() : bool
	GetCapSpeedStepsBackward() : int
	GetCapSpeedStepsForward() : int
	GetCapStatisticsReporting() : bool
	GetCapUpdateFirmware() : bool
	GetCapUpdateStatistics() : bool
	GetCheckHealthText() : string
	GetClaimed() : bool
	GetDeviceControlDescription() : string
	GetDeviceControlVersion() : UposVersion
	GetDeviceEnabled() : bool
	GetDeviceServiceDescription() : string
	GetDeviceServiceVersion() : UposVersion
	GetFreezeEvents() : bool
	GetLightBarrierBackwardInterrupted() : bool
	GetLightBarrierForwardInterrupted() : bool
	GetMotionStatus() : BeltMotionStatus
	GetPhysicalDeviceDescription() : string
	GetPhysicalDeviceName() : string
	GetPowerNotify() : PowerNotification

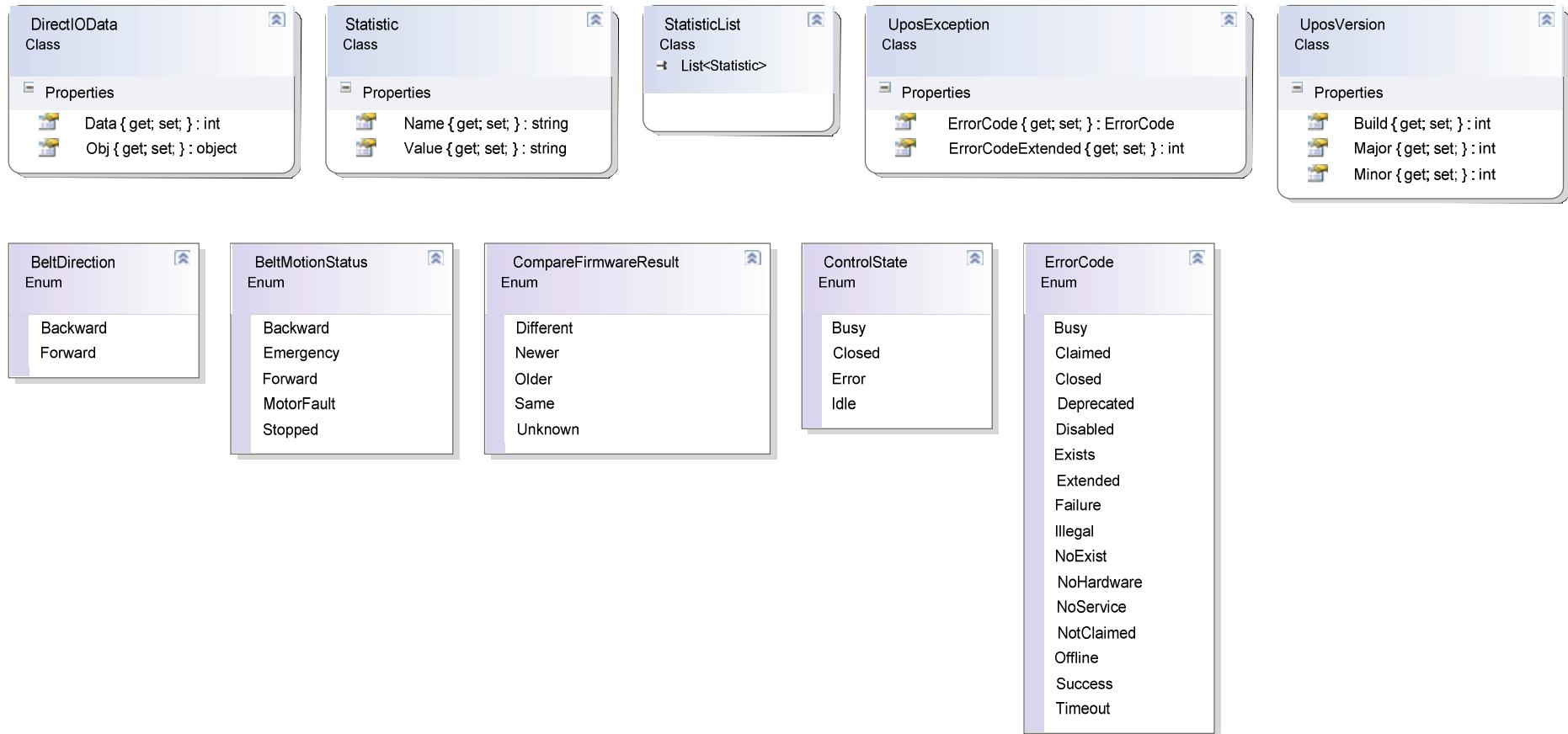
Belt



Methods

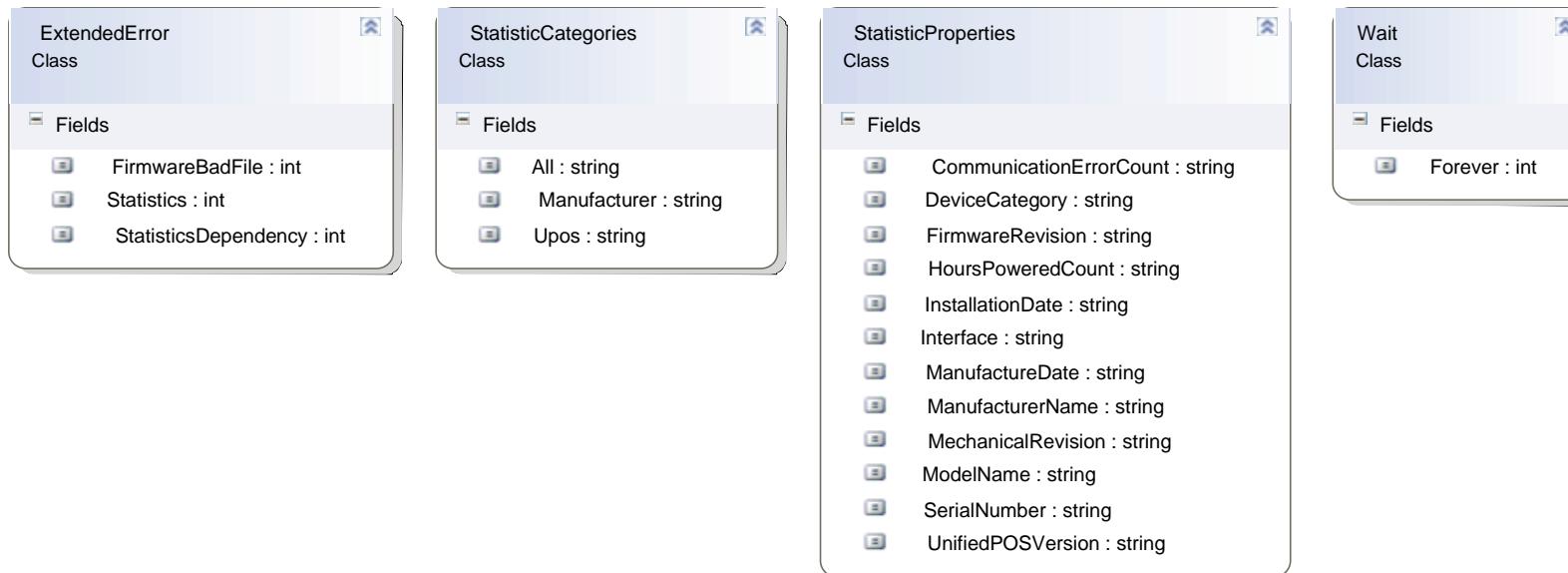
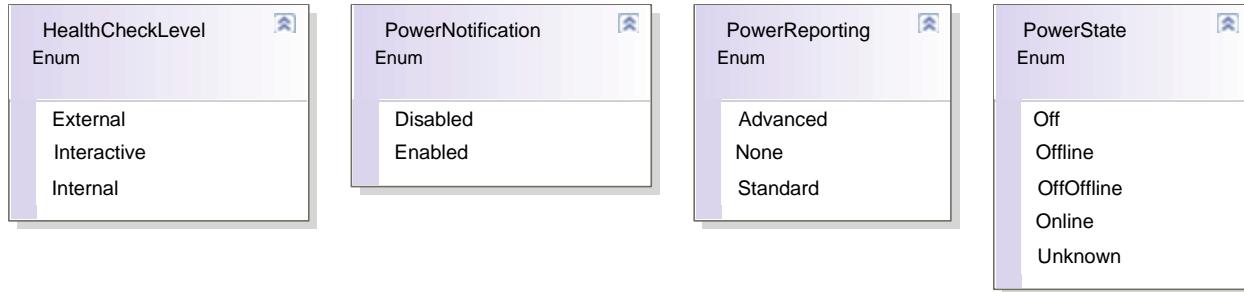
- » GetPowerState() : PowerState
- » GetSecurityFlapBackwardOpened() : bool
- » GetSecurityFlapForwardOpened() : bool
- » GetState() : ControlState
- » MoveBackward(int Speed) : void
- » MoveForward(int Speed) : void
- » Open(string EndpointAddress) : void
- » Release() : void
- » ResetBelt() : void
- » ResetItemCount(BeltDirection Direction) : void
- » ResetStatistics(StatisticList StatisticsBuffer) : void
- » RetrieveStatistics(StatisticList StatisticsBuffer) : string
- » SetAutoStopBackward(bool AutoStopBackward) : void
 - » SetAutoStopBackwardDelayTime(int AutoStopBackwardDelayTime) : void
- » SetAutoStopForward(bool AutoStopForward) : void
 - » SetAutoStopForwardDelayTime(int AutoStopForwardDelayTime) : void
- » SetDeviceEnabled(bool DeviceEnabled) : void
- » SetFreezeEvents(bool FreezeEvents) : void
- » SetPowerNotify(PowerNotification PowerNotify) : void
- » StopBelt() : void
- » UpdateFirmware(string FirmwareFileName) : void
- » UpdateStatistics(StatisticList StatisticsBuffer) : void

Belt

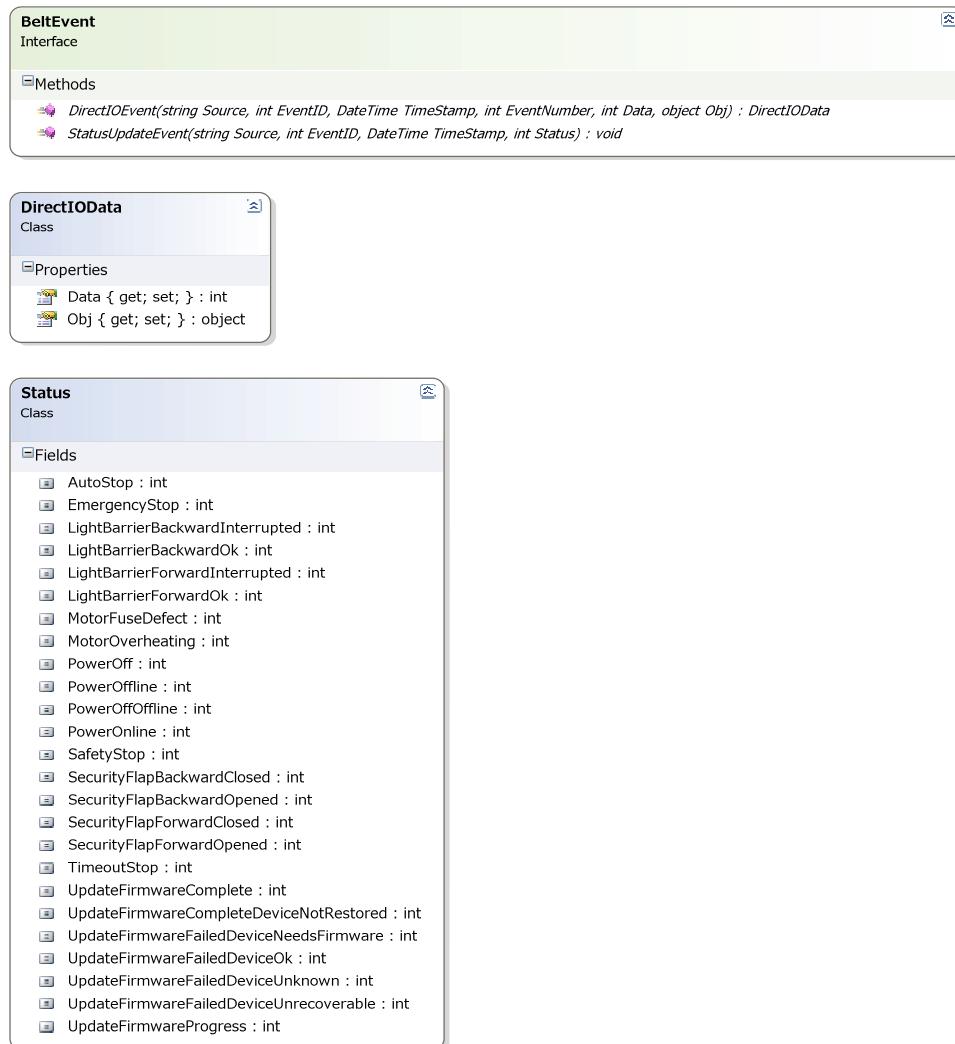


WS-POS 1.1 Technical Specification

Belt



Belt Events

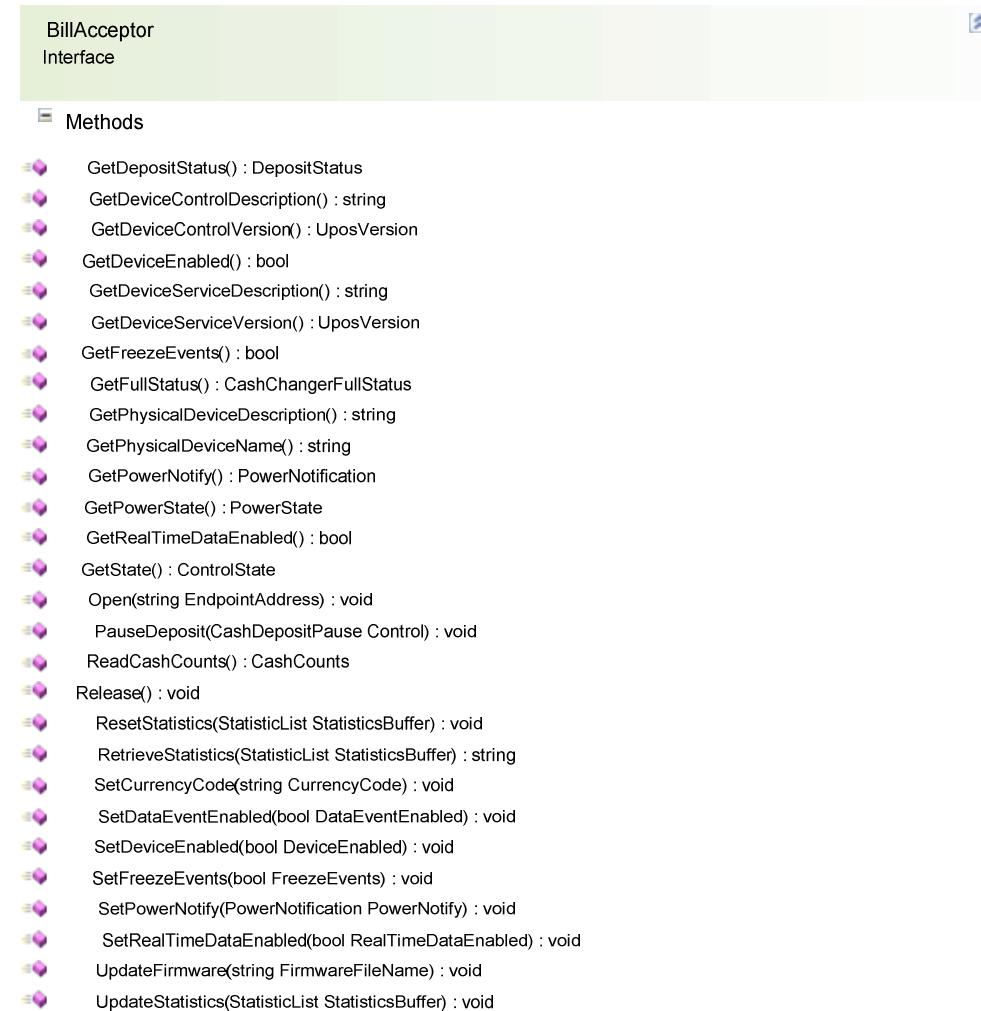


Bill Acceptor

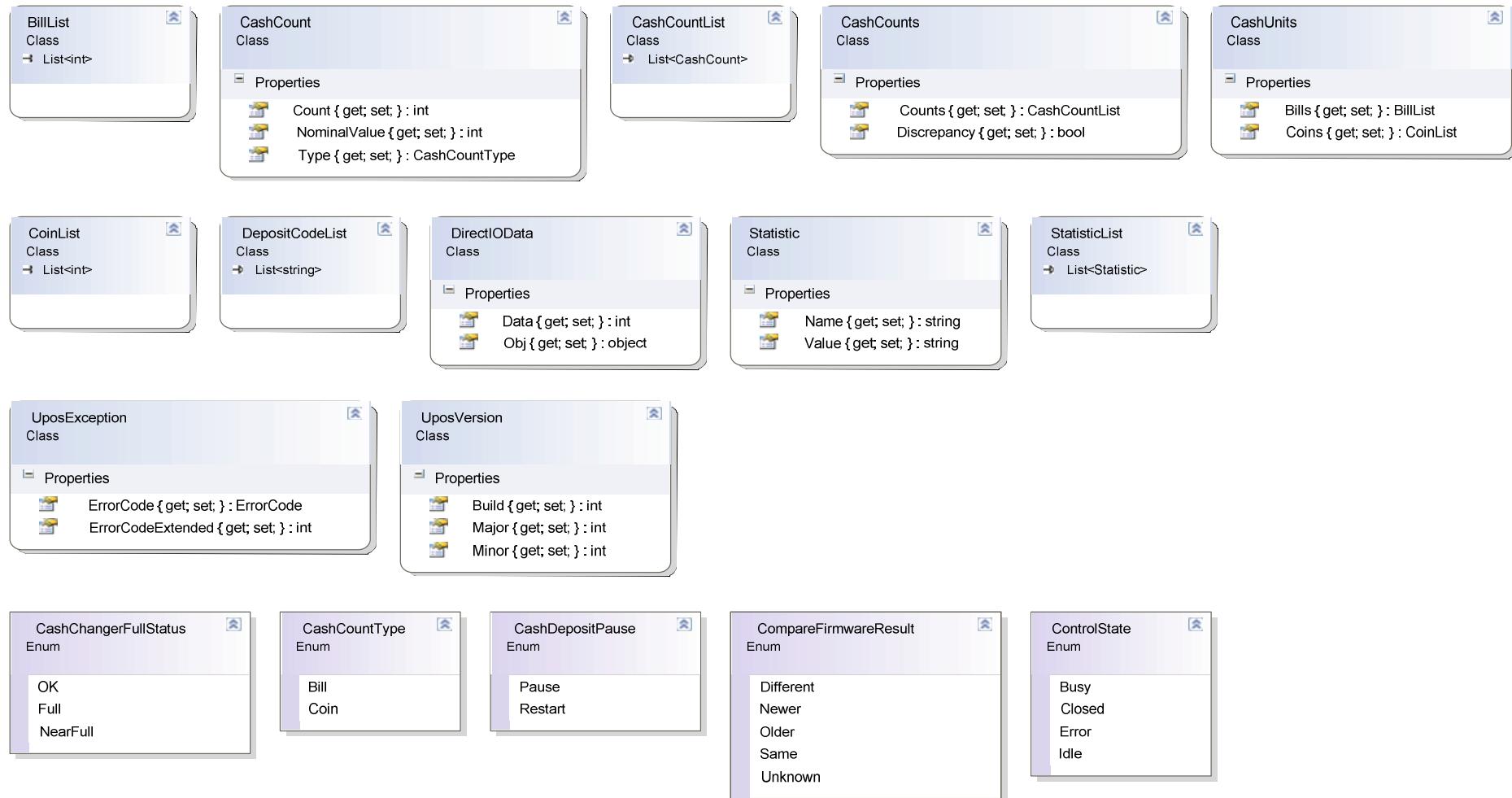
The screenshot shows a software interface with a light green header bar containing the text "BillAcceptor Interface". Below this is a white main area. On the left side of the main area, there is a tree view with a single node labeled "Methods" which is expanded, showing a list of method names. Each method name is preceded by a small purple icon. The methods listed are:

- AdjustCashCounts(CashCountList CashCounts) : void
- BeginDeposit() : void
- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- ClearInput() : void
- Close(string EndpointAddress) : void
 - CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
 - DirectIO(int Command, int Data, object Obj) : DirectIOData
- EndDeposit(EndDepositAction Success) : void
- FixDeposit() : void
- GetCapCompareFirmwareVersion() : bool
- GetCapDiscrepancy() : bool
- GetCapFullSensor() : bool
- GetCapJamSensor() : bool
- GetCapNearFullSensor() : bool
- GetCapPauseDeposit() : bool
- GetCapPowerReporting() : PowerReporting
- GetCapRealTimeData() : bool
- GetCapStatisticsReporting() : bool
- GetCapUpdateFirmware() : bool
- GetCapUpdateStatistics() : bool
- GetCheckHealthText() : string
- GetClaimed() : bool
- GetCurrencyCode() : string
- GetDataCount() : int
- GetDataEventEnabled() : bool
- GetDepositAmount() : int
- GetDepositCashList() : CashUnits
- GetDepositCodeList() : DepositCodeList
- GetDepositCounts() : CashCountList

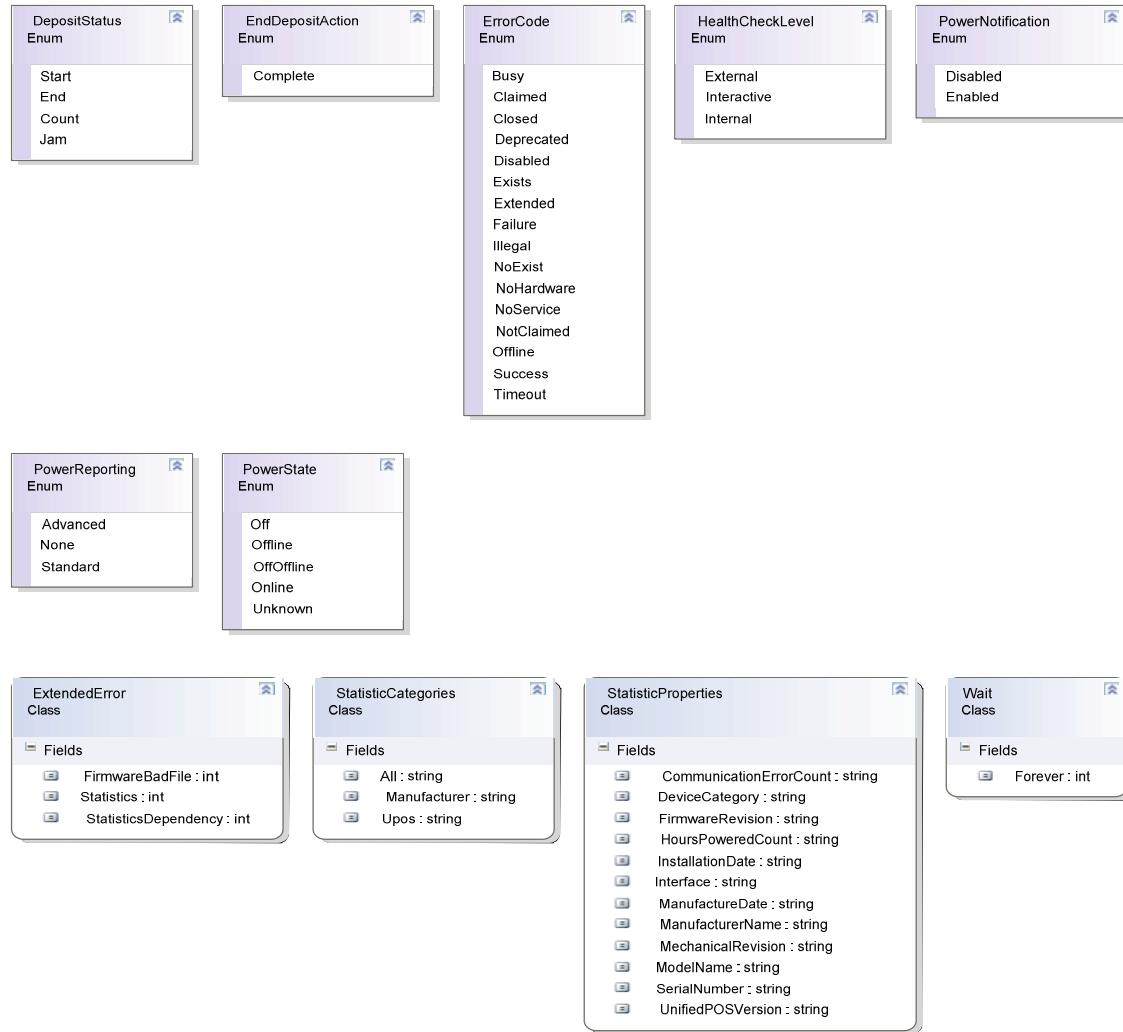
Bill Acceptor



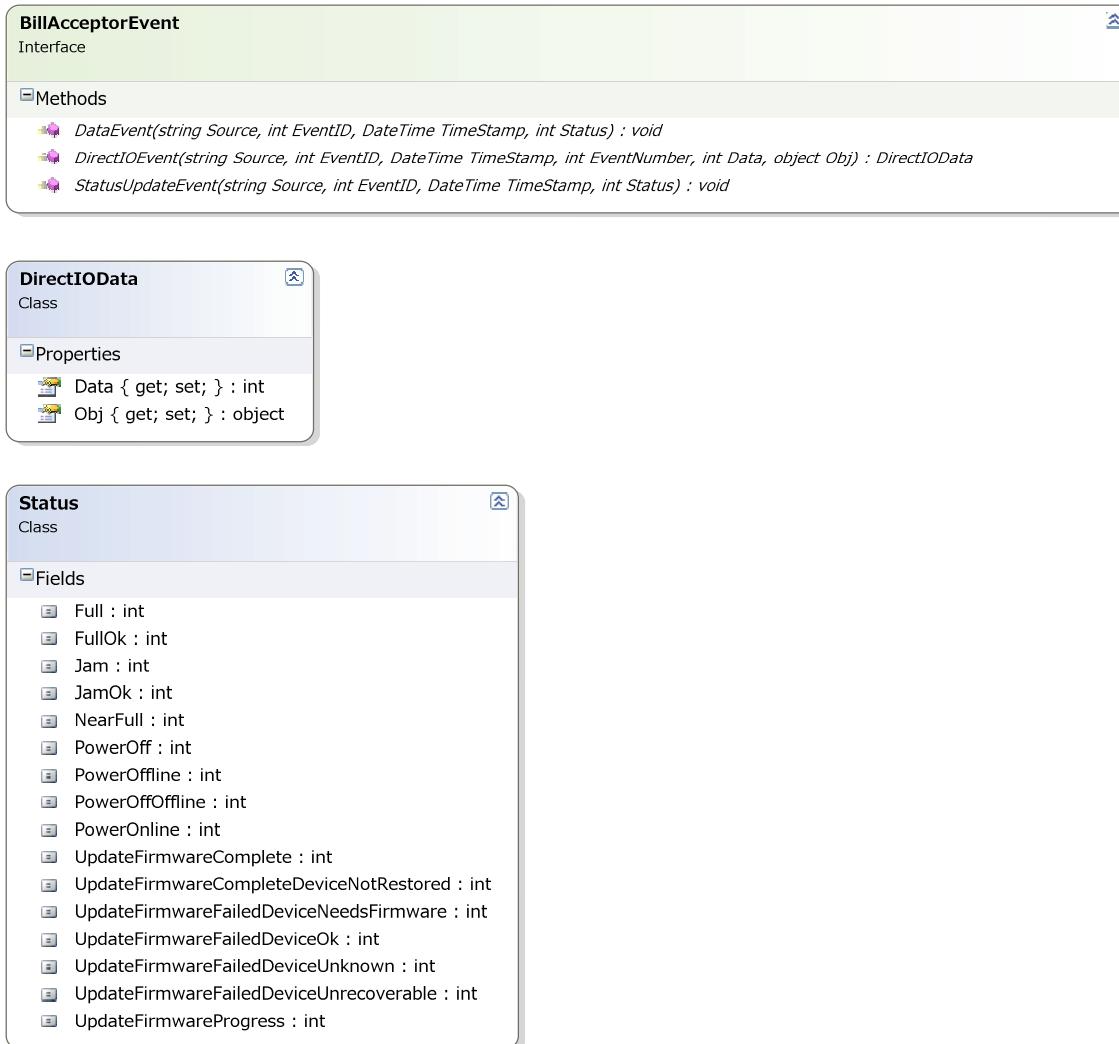
Bill Acceptor



Bill Acceptor



Bill Acceptor Events



Bill Dispenser

The diagram shows a UML class named "BillDispenser Interface". It has a single association named "Methods" which contains a list of methods. Each method is preceded by a small purple icon.

- Methods
 - AdjustCashCounts(CashCountList CashCounts) : void
 - CheckHealth(HealthCheckLevel Level) : void
 - Claim(int Timeout) : void
 - Close(string EndpointAddress) : void
 - CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
 - DirectIO(int Command, int Data, object Obj) : DirectIOData
 - DispenseCash(CashCountList CashCounts) : void
 - GetAsyncMode() : bool
 - GetAsyncResultCode() : int
 - GetAsyncResultCodeExtended() : int
 - GetCapCompareFirmwareVersion() : bool
 - GetCapDiscrepancy() : bool
 - GetCapEmptySensor() : bool
 - GetCapJamSensor() : bool
 - GetCapNearEmptySensor() : bool
 - GetCapPowerReporting() : PowerReporting
 - GetCapStatisticsReporting() : bool
 - GetCapUpdateFirmware() : bool
 - GetCapUpdateStatistics() : bool
 - GetCheckHealthText() : string
 - GetClaimed() : bool
 - GetCurrencyCashList() : CashUnits
 - GetCurrencyCode() : string
 - GetCurrencyCodeList() : CurrencyCodeList
 - GetCurrentExit() : int
 - GetDeviceControlDescription() : string
 - GetDeviceControlVersion() : UposVersion
 - GetDeviceEnabled() : bool
 - GetDeviceExists() : int

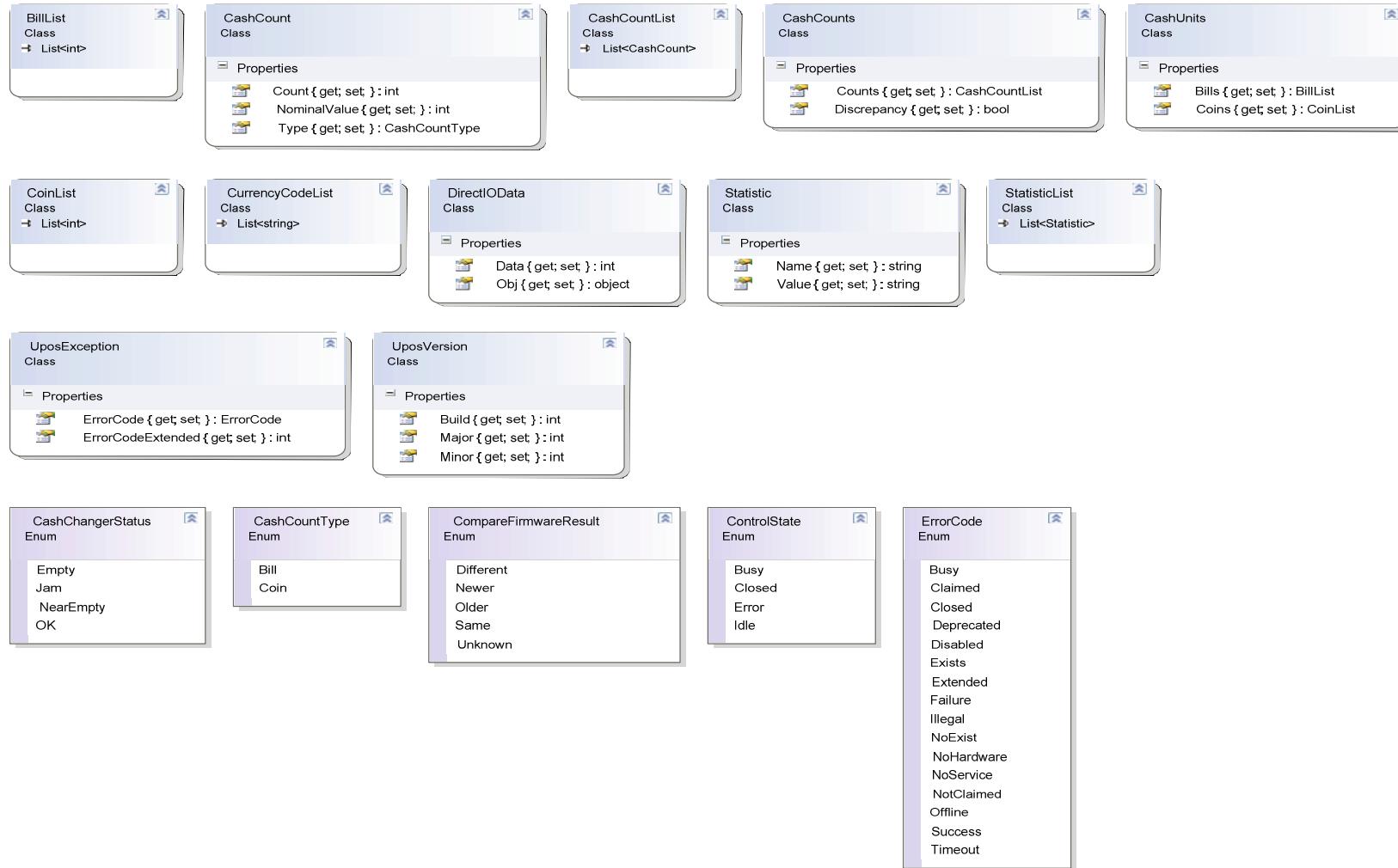
Bill Dispenser

The screenshot shows a software interface for the Bill Dispenser. At the top left, it says "BillDispenser Interface". Below that is a tree view under the heading "Methods". The methods listed are:

- GetDeviceServiceDescription() : string
- GetDeviceServiceVersion() : UposVersion
- GetDeviceStatus() : CashChangerStatus
- GetExitCashList() : CashUnits
- GetFreezeEvents() : bool
- GetPhysicalDeviceDescription() : string
- GetPhysicalDeviceName() : string
- GetPowerNotify() : PowerNotification
- GetPowerState() : PowerState
- GetState() : ControlState
- Open(string EndpointAddress) : void
- ReadCashCounts() : CashCounts
- Release() : void
- ResetStatistics(StatisticList StatisticsBuffer) : void
- RetrieveStatistics(StatisticList StatisticsBuffer) : string
- SetAsyncMode(bool AsyncMode) : void
- SetCurrencyCode(string CurrencyCode) : void
- SetCurrentExit(int CurrentExit) : void
- SetDeviceEnabled(bool DeviceEnabled) : void
- SetFreezeEvents(bool FreezeEvents) : void
- SetPowerNotify(PowerNotification PowerNotify) : void
- UpdateFirmware(string FirmwareFileName) : void
- UpdateStatistics(StatisticList StatisticsBuffer) : void

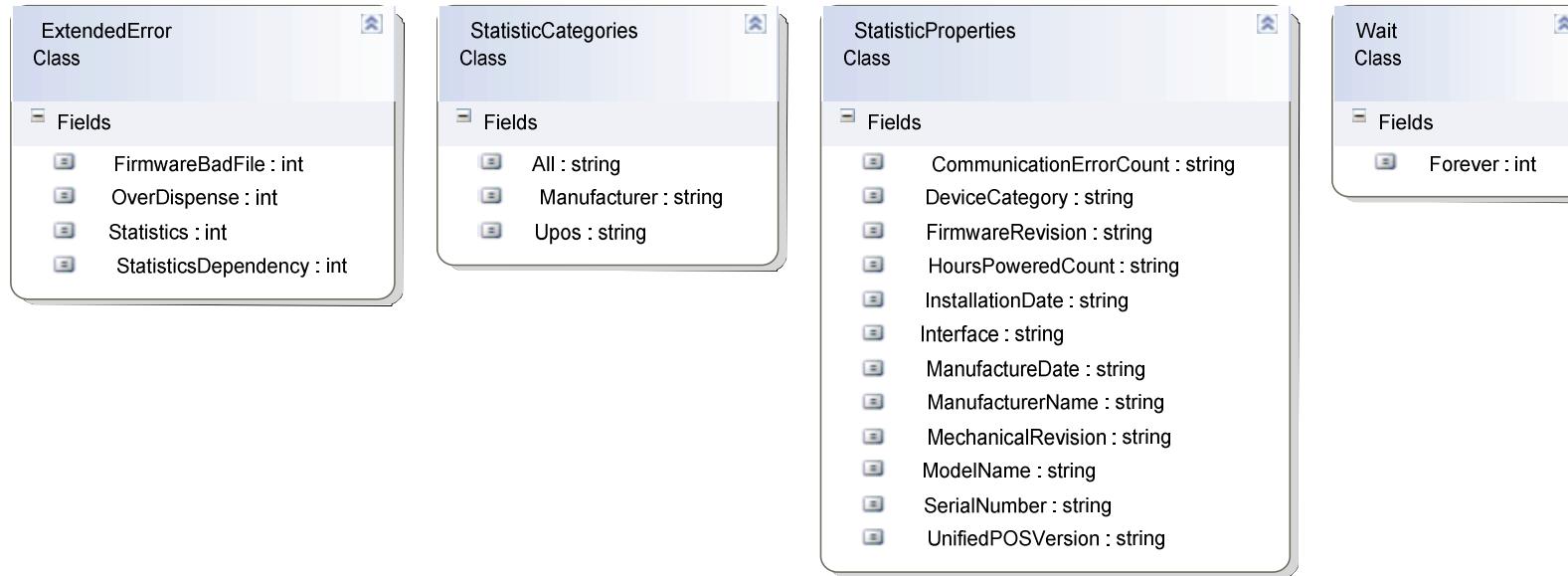
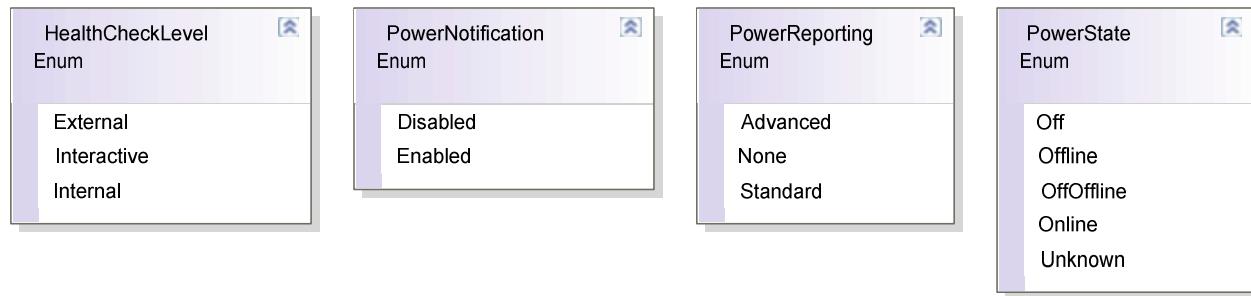
WS-POS 1.1 Technical Specification

Bill Dispenser

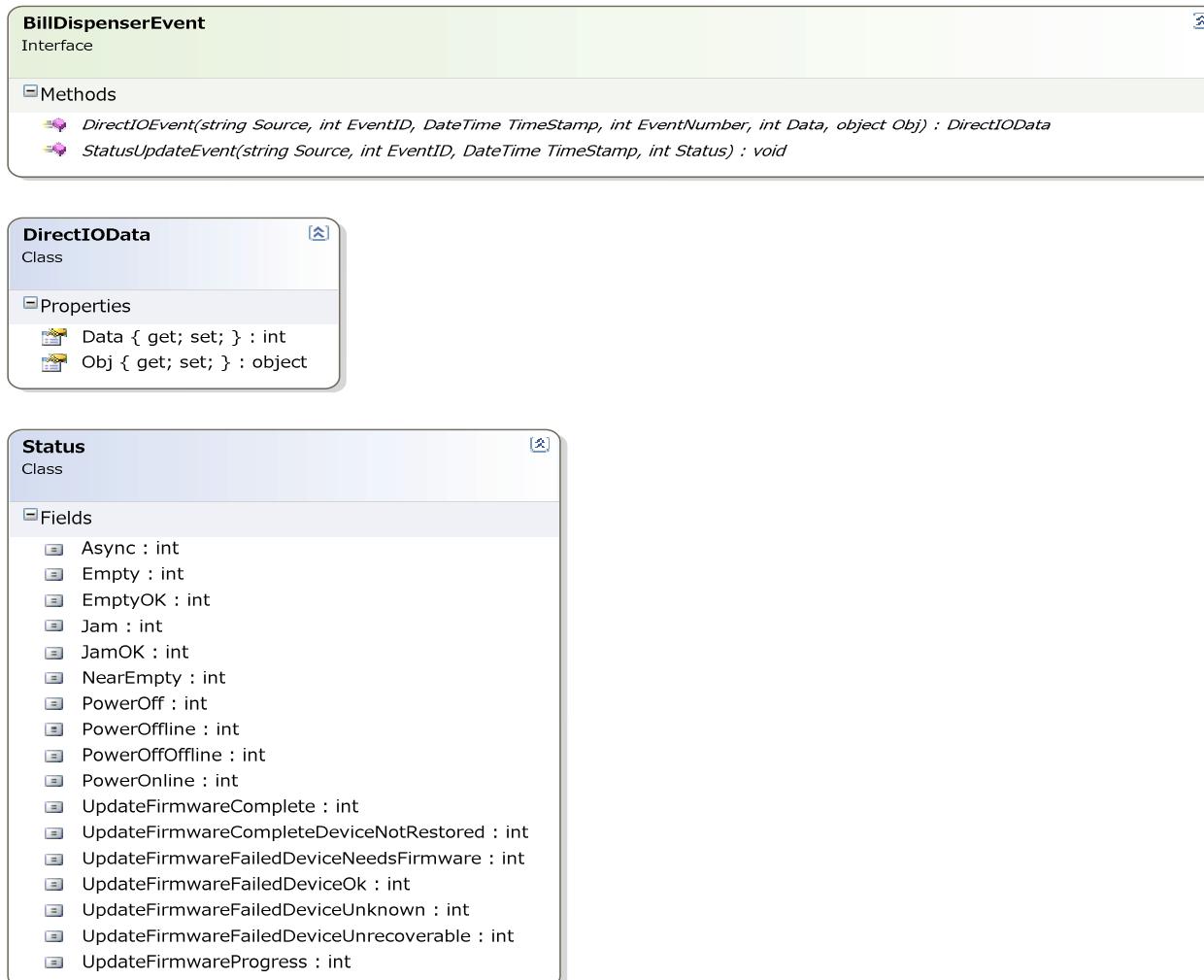


WS-POS 1.1 Technical Specification

Bill Dispenser



Bill Dispenser Events



Biometrics

The diagram shows a UML class named "Biometrics Interface". It has a single association named "Methods" which contains a list of method signatures. The methods include:

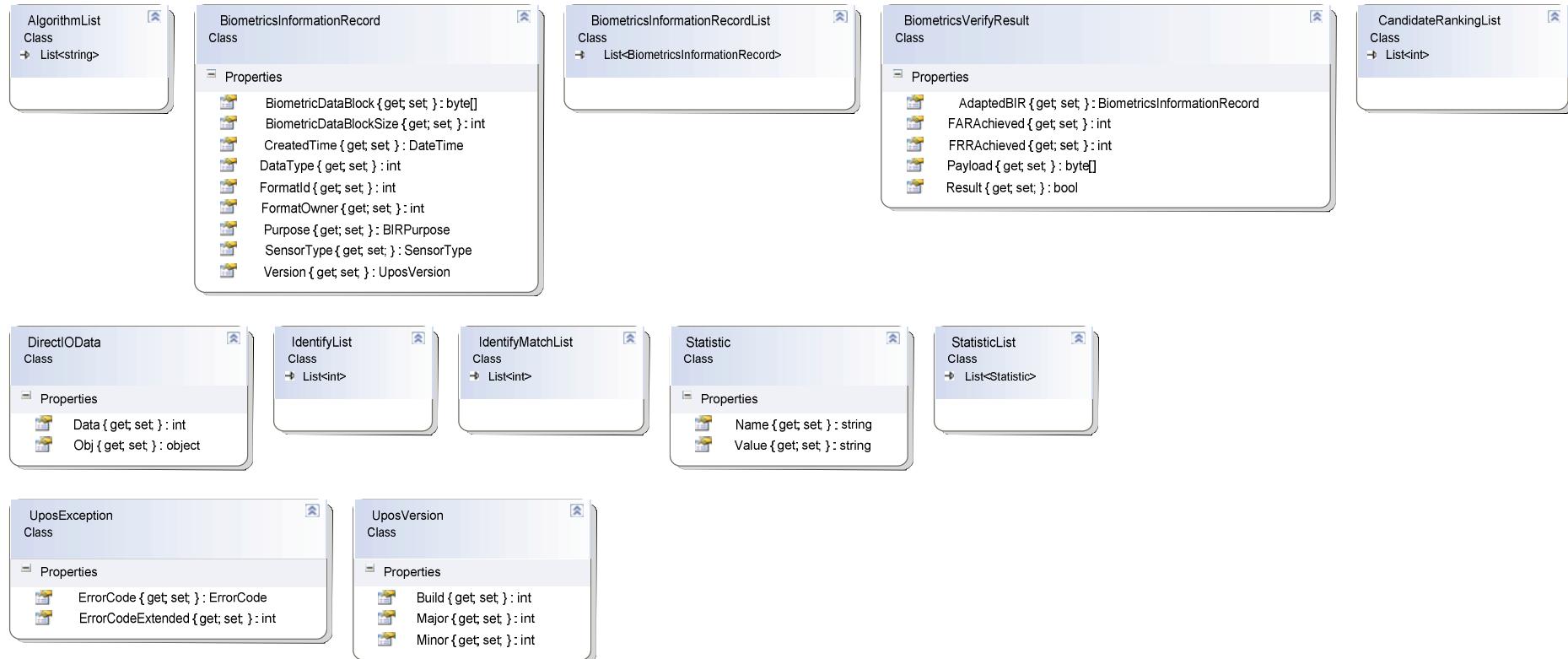
- BeginEnrollCapture(BiometricsInformationRecord ReferenceBIR, byte[] Payload) : void
- BeginVerifyCapture() : void
- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- ClearInput() : void
- ClearInputProperties() : void
- Close(string EndpointAddress) : void
- CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
- DirectIO(int Command, int Data, object Obj) : DirectIOData
- EndCapture() : void
- GetAlgorithm() : int
- GetAlgorithmList() : AlgorithmList
- GetAutoDisable() : bool
- GetBIR() : BiometricsInformationRecord
- GetCapCompareFirmwareVersion() : bool
- GetCapPowerReporting() : PowerReporting
- GetCapPrematchData() : bool
- GetCapRawSensorData() : bool
- GetCapRealTimeData() : bool
- GetCapSensorColor() : int
- GetCapSensorOrientation() : int
- GetCapSensorType() : int
- GetCapStatisticsReporting() : bool
- GetCapTemplateAdaptation() : bool
- GetCapUpdateFirmware() : bool
- GetCapUpdateStatistics() : bool
- GetCheckHealthText() : string
- GetClaimed() : bool
- GetDataCount() : int
- GetDataEventEnabled() : bool
- GetDeviceControlDescription() : string
- GetDeviceControlVersion() : UposVersion
- GetDeviceEnabled() : bool
- GetDeviceServiceDescription() : string
- GetDeviceServiceVersion() : UposVersion
- GetFreezeEvents() : bool
- GetPhysicalDeviceDescription() : string
- GetPhysicalDeviceName() : string
- GetPowerNotify() : PowerNotification

Biometrics

Biometrics Interface	
Methods	
<ul style="list-style-type: none">≡ GetPowerState() : PowerState≡ GetRawSensorData() : byte[]≡ GetRealTimeDataEnabled() : bool≡ GetSensorBPP() : int≡ GetSensorColor() : SensorColor≡ GetSensorHeight() : int≡ GetSensorOrientation() : SensorOrientation≡ GetSensorType() : SensorType≡ GetSensorWidth() : int≡ GetState() : ControlState≡ Identify(int MaxFARRequested, int MaxFRRRequested, bool FARPrecedence, BiometricsInformationRecordList ReferenceBIRPopulation , int Timeout) : IdentifyList≡ IdentifyMatch(int MaxFARRequested, int MaxFRRRequested, bool FARPrecedence, BiometricsInformationRecord SampleBIR, Biometrics InformationRecordList ReferenceBIRPopulation) : IdentifyMatchList≡ Open(string EndpointAddress) : void≡ ProcessPrematchData(BiometricsInformationRecord SampleBIR, BiometricsInformationRecord PrematchDataBIR) : BiometricsInformationRecord≡ Release() : void≡ ResetStatistics(StatisticList StatisticsBuffer) : void≡ RetrieveStatistics(StatisticList StatisticsBuffer) : string≡ SetAlgorithm(int Algorithm) : void≡ SetAutoDisable(bool AutoDisable) : void≡ SetDataEventEnabled(bool DataEventEnabled) : void≡ SetDeviceEnabled(bool DeviceEnabled) : void≡ SetFreezeEvents(bool FreezeEvents) : void≡ SetPowerNotify(PowerNotification PowerNotify) : void≡ SetRealTimeDataEnabled(bool RealTimeDataEnabled) : void≡ SetSensorColor(SensorColor SensorColor) : void≡ SetSensorOrientation(SensorOrientation SensorOrientation) : void≡ SetSensorType(SensorType SensorType) : void≡ UpdateFirmware(string FirmwareFileName) : void≡ UpdateStatistics(StatisticList StatisticsBuffer) : void≡ Verify(int MaxFARRequested, int MaxFRRRequested, bool FARPrecedence, BiometricsInformationRecord ReferenceBIR, bool AdaptBIR, int Timeout) : BiometricsVerifyResult≡ VerifyMatch(int MaxFARRequested, int MaxFRRRequested, bool FARPrecedence, BiometricsInformationRecord SampleBIR, BiometricsInformationRecord ReferenceBIR, bool AdaptBIR) : BiometricsVerifyResult	

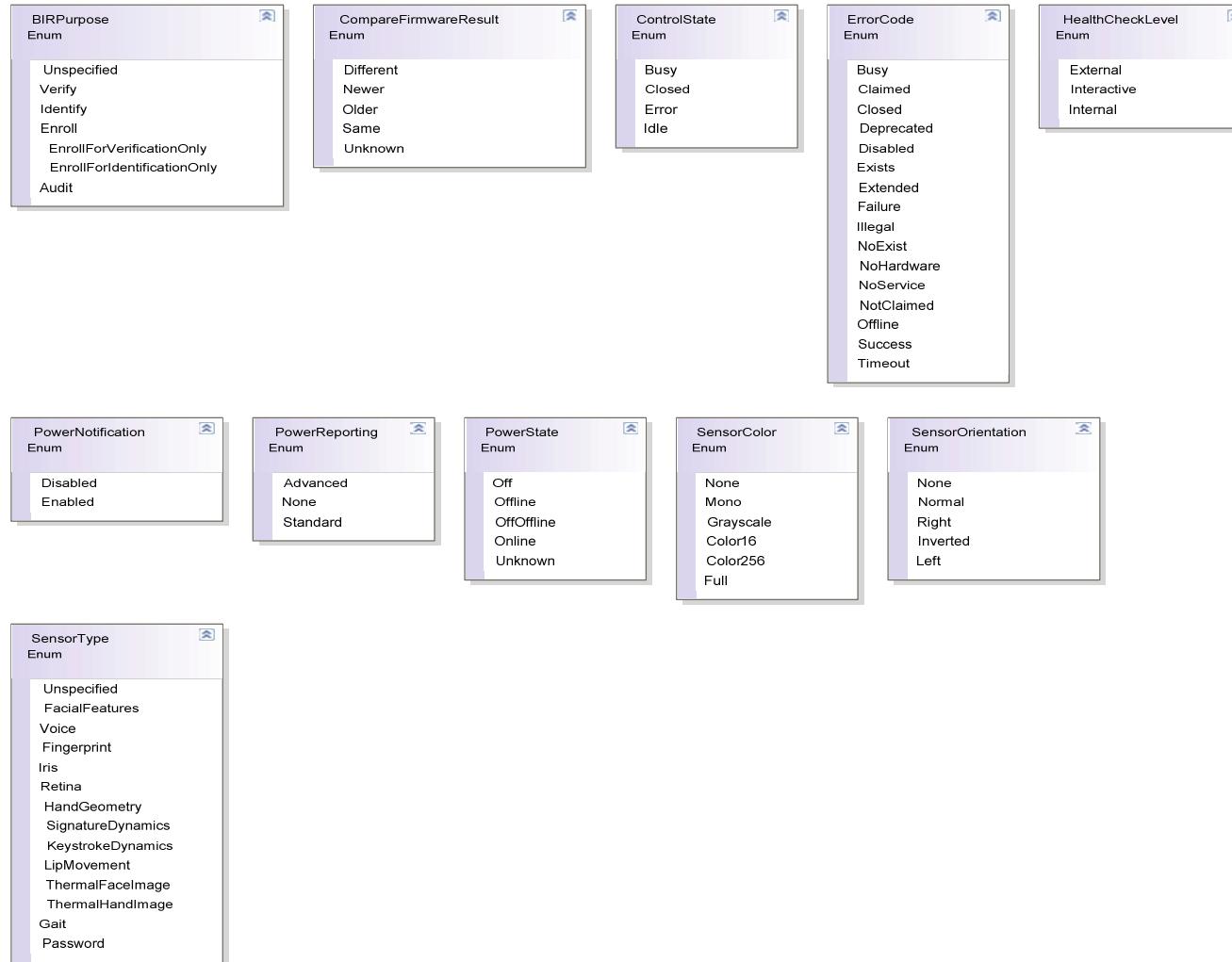
WS-POS 1.1 Technical Specification

Biometrics



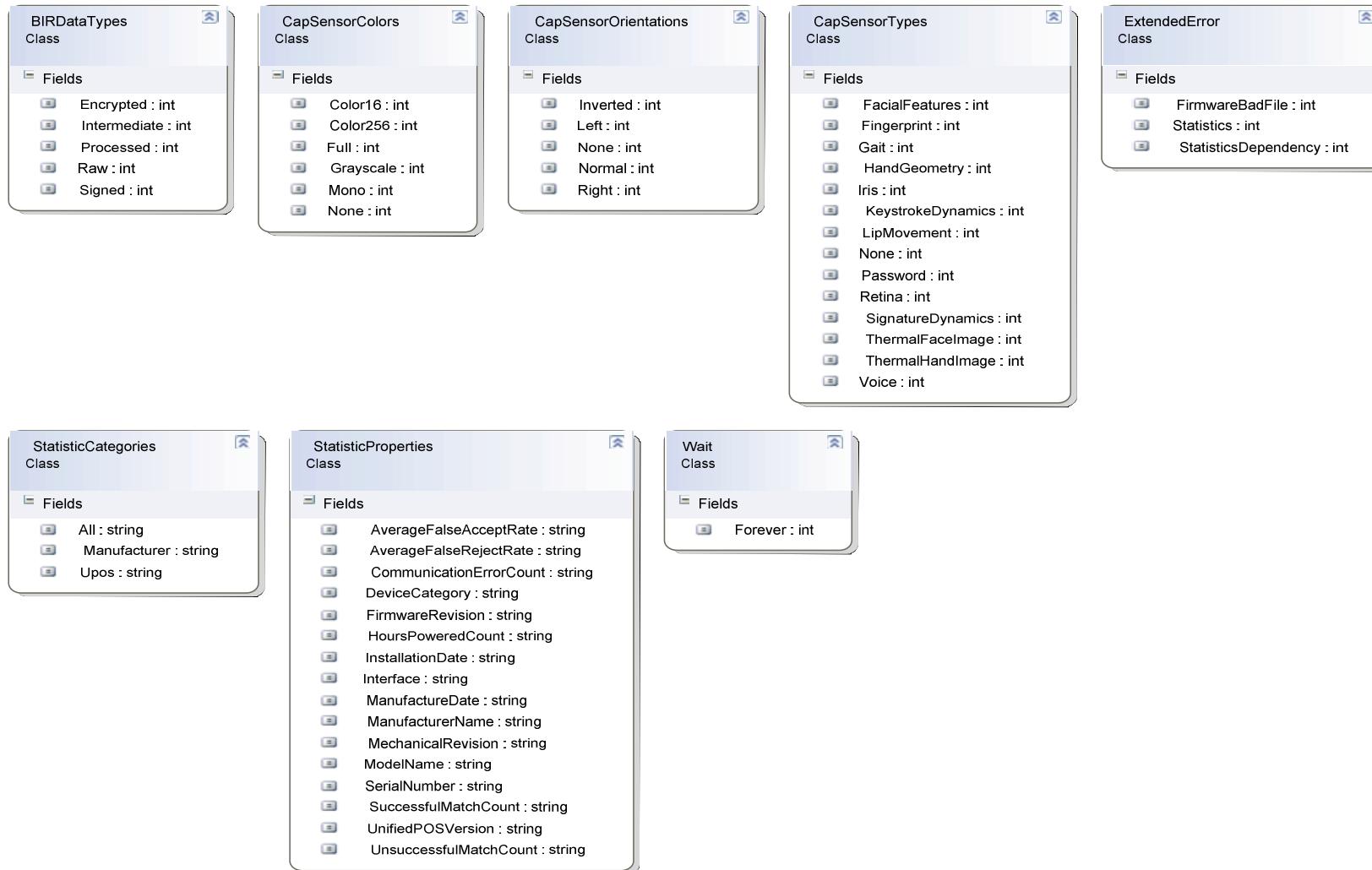
WS-POS 1.1 Technical Specification

Biometrics

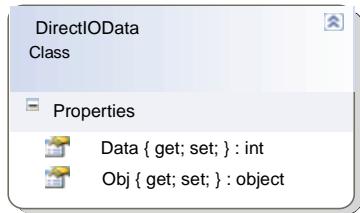
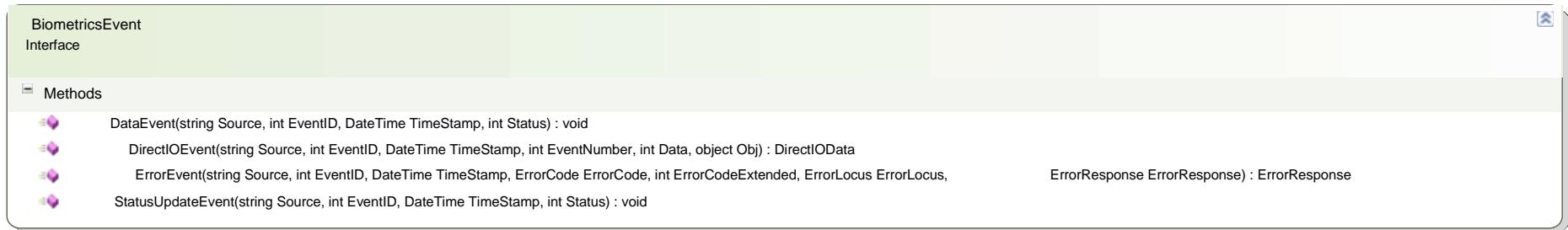


WS-POS 1.1 Technical Specification

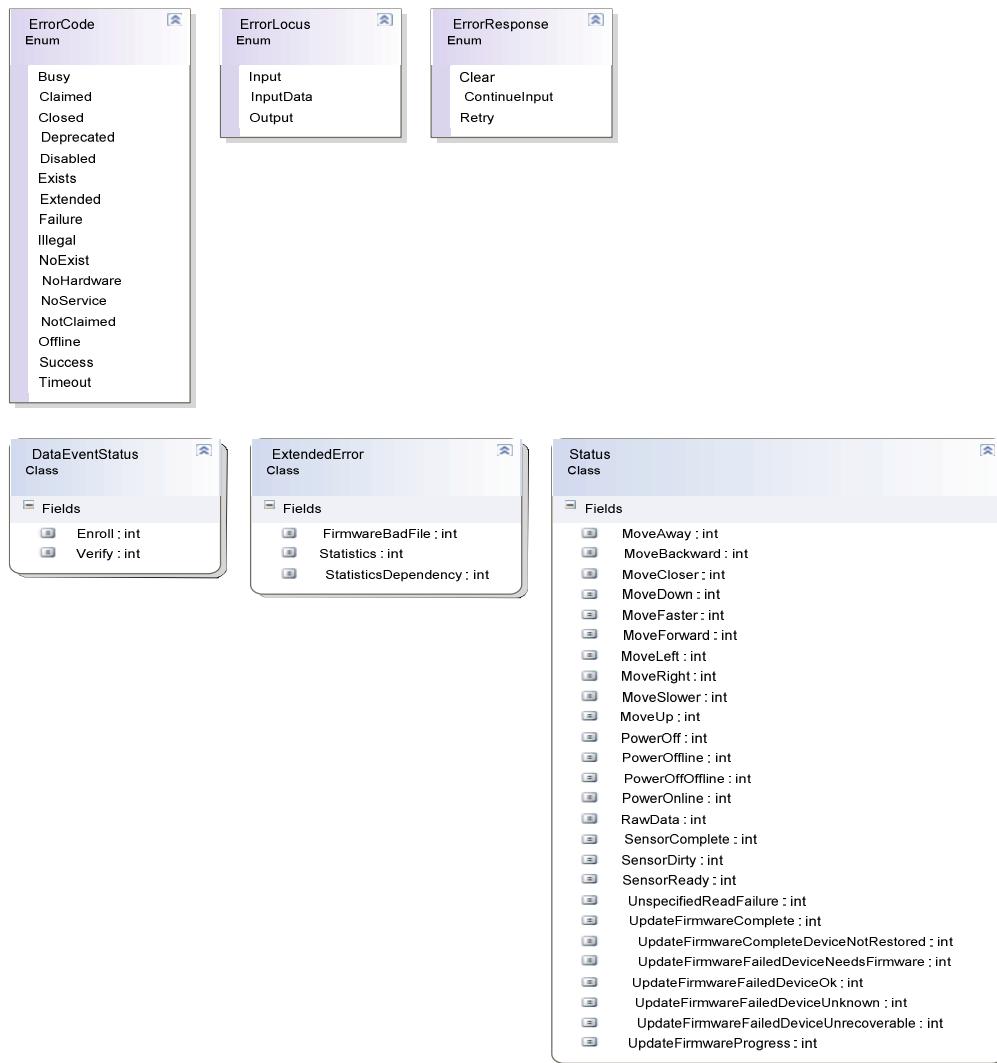
Biometrics



Biometrics Events



Biometric Events



Bump Bar

The diagram shows the **BumpBar** interface with its associated methods. It includes associations with **HealthCheckLevel**, **EndpointAddress**, **PowerReporting**, **UposVersion**, and **CompareFirmwareResult**.

BumpBar Interface

Methods

- BumpBarSound(int Units, int Frequency, int Duration, int NumberOfCycles, int InterSoundWait) : void
- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- ClearInput() : void
- ClearOutput() : void
- Close(string EndpointAddress) : void
 - CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
 - DirectIO(int Command, int Data, object Obj) : DirectIOData
- GetAsyncMode() : bool
- GetAutoToneDuration() : int
- GetAutoToneFrequency() : int
- GetBumpBarDataCount() : int
- GetCapCompareFirmwareVersion() : bool
- GetCapPowerReporting() : PowerReporting
- GetCapStatisticsReporting() : bool
- GetCapTone() : bool
- GetCapUpdateFirmware() : bool
- GetCapUpdateStatistics() : bool
- GetCheckHealthText() : string
- GetClaimed() : bool
- GetCurrentUnitID() : int
- GetDataCount() : int
- GetDataEventEnabled() : bool
- GetDeviceControlDescription() : string
- GetDeviceControlVersion() : UposVersion
- GetDeviceEnabled() : bool
- GetDeviceServiceDescription() : string
- GetDeviceServiceVersion() : UposVersion
- GetErrorString() : string
- GetErrorUnits() : int
- GetEventString() : string

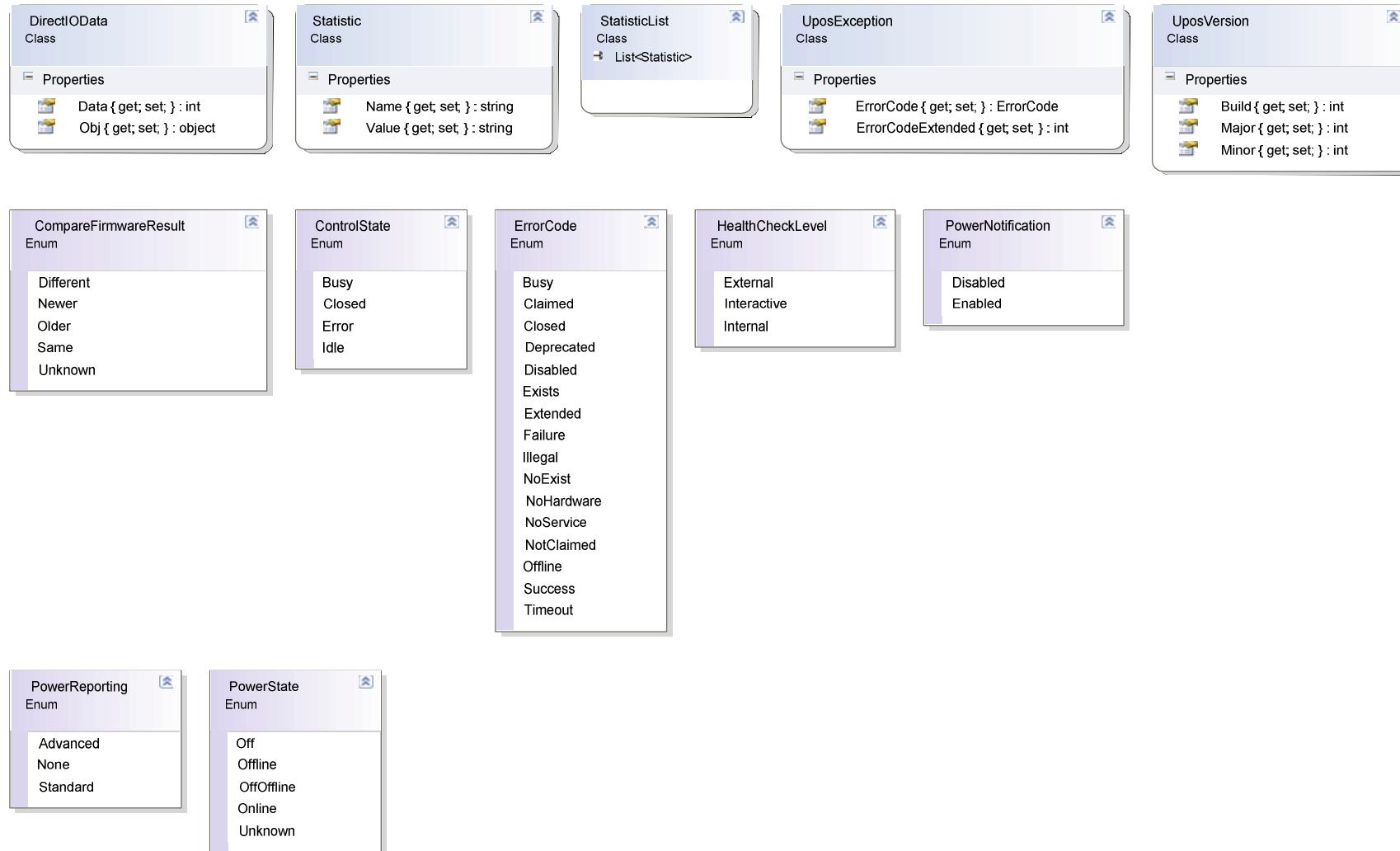
Bump Bar



- ⊕ GetEventUnitID() : int
- ⊕ GetEventUnits() : int
- ⊕ GetFreezeEvents() : bool
- ⊕ GetKeys() : int
- ⊕ GetOutputID() : int
- ⊕ GetPhysicalDeviceDescription() : string
- ⊕ GetPhysicalDeviceName() : string
- ⊕ GetPowerNotify() : PowerNotification
- ⊕ GetPowerState() : PowerState
- ⊕ GetState() : ControlState
- ⊕ GetTimeout() : int
- ⊕ GetUnitsOnline() : int
- ⊕ Open(string EndpointAddress) : void
- ⊕ Release() : void
- ⊕ ResetStatistics(StatisticList StatisticsBuffer) : void
- ⊕ RetrieveStatistics(StatisticList StatisticsBuffer) : string
- ⊕ SetAsyncMode(bool AsyncMode) : void
- ⊕ SetAutoToneDuration(int AutoToneDuration) : void
- ⊕ SetAutoToneFrequency(int AutoToneFrequency) : void
- ⊕ SetCurrentUnitID(int CurrentUnitID) : void
- ⊕ SetDataEventEnabled(bool DataEventEnabled) : void
- ⊕ SetDeviceEnabled(bool DeviceEnabled) : void
- ⊕ SetFreezeEvents(bool FreezeEvents) : void
- ⊕ SetKeyTranslation(int Units, int ScanCode, int LogicalKey) : void
- ⊕ SetPowerNotify(PowerNotification PowerNotify) : void
- ⊕ SetTimeout(int Timeout) : void
- ⊕ UpdateFirmware(string FirmwareFileName) : void
- ⊕ UpdateStatistics(StatisticList StatisticsBuffer) : void

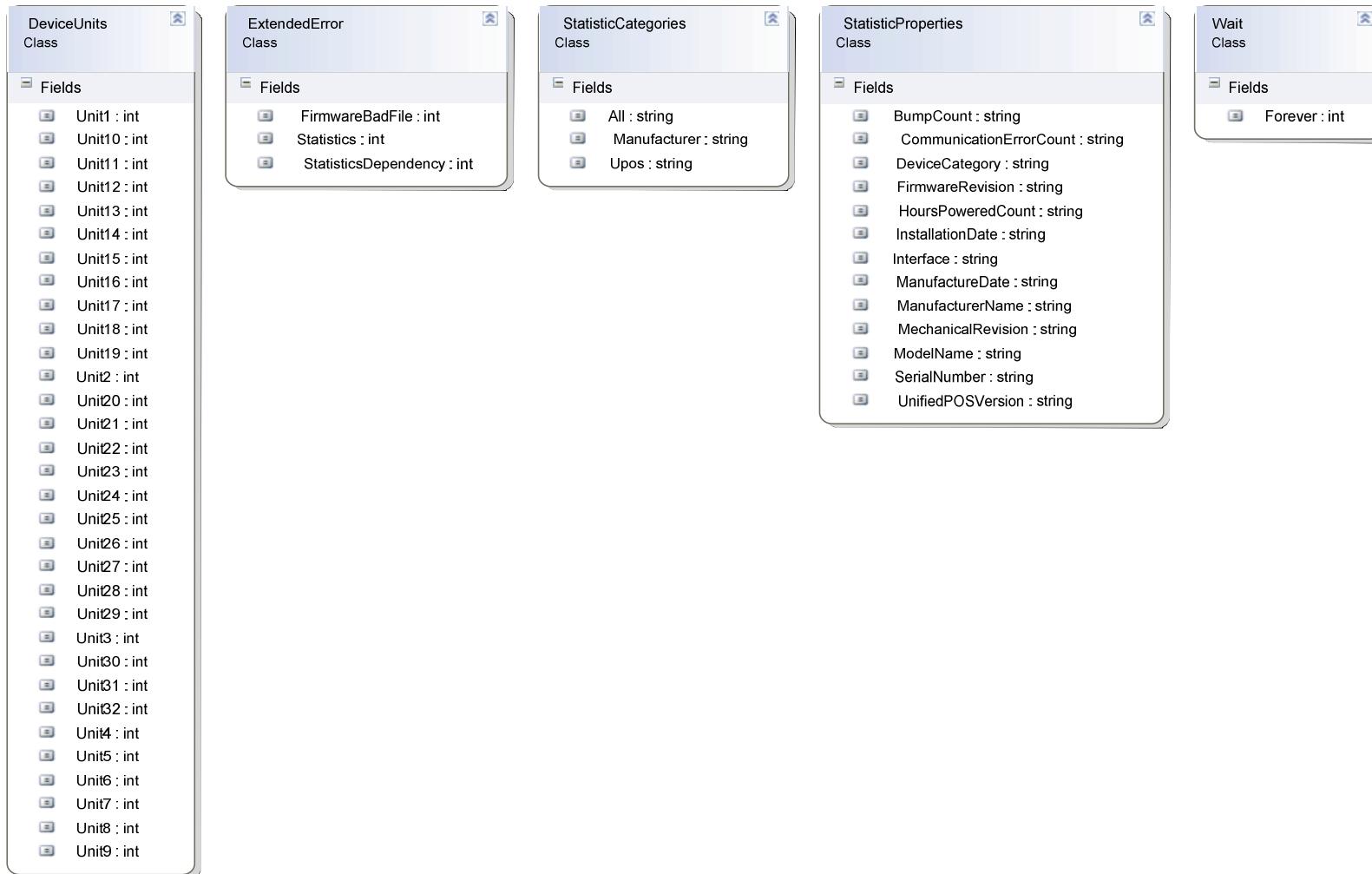
WS-POS 1.1 Technical Specification

Bump Bar

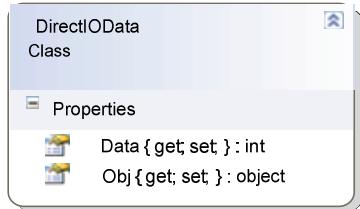
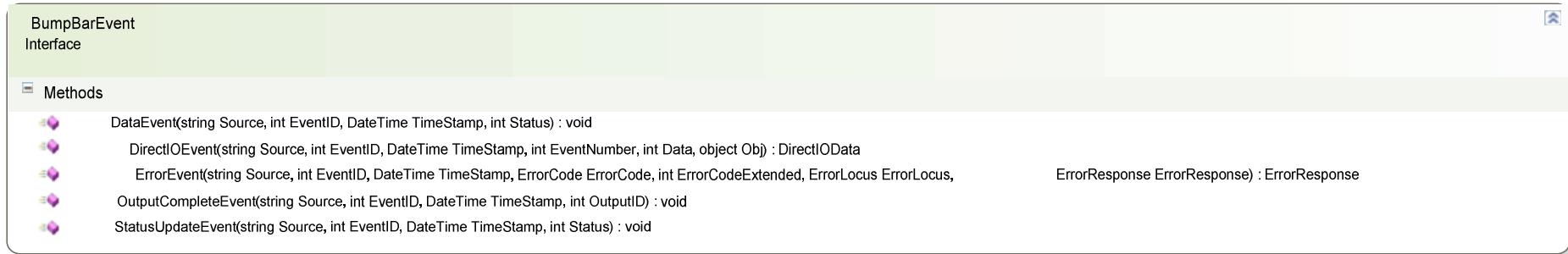


WS-POS 1.1 Technical Specification

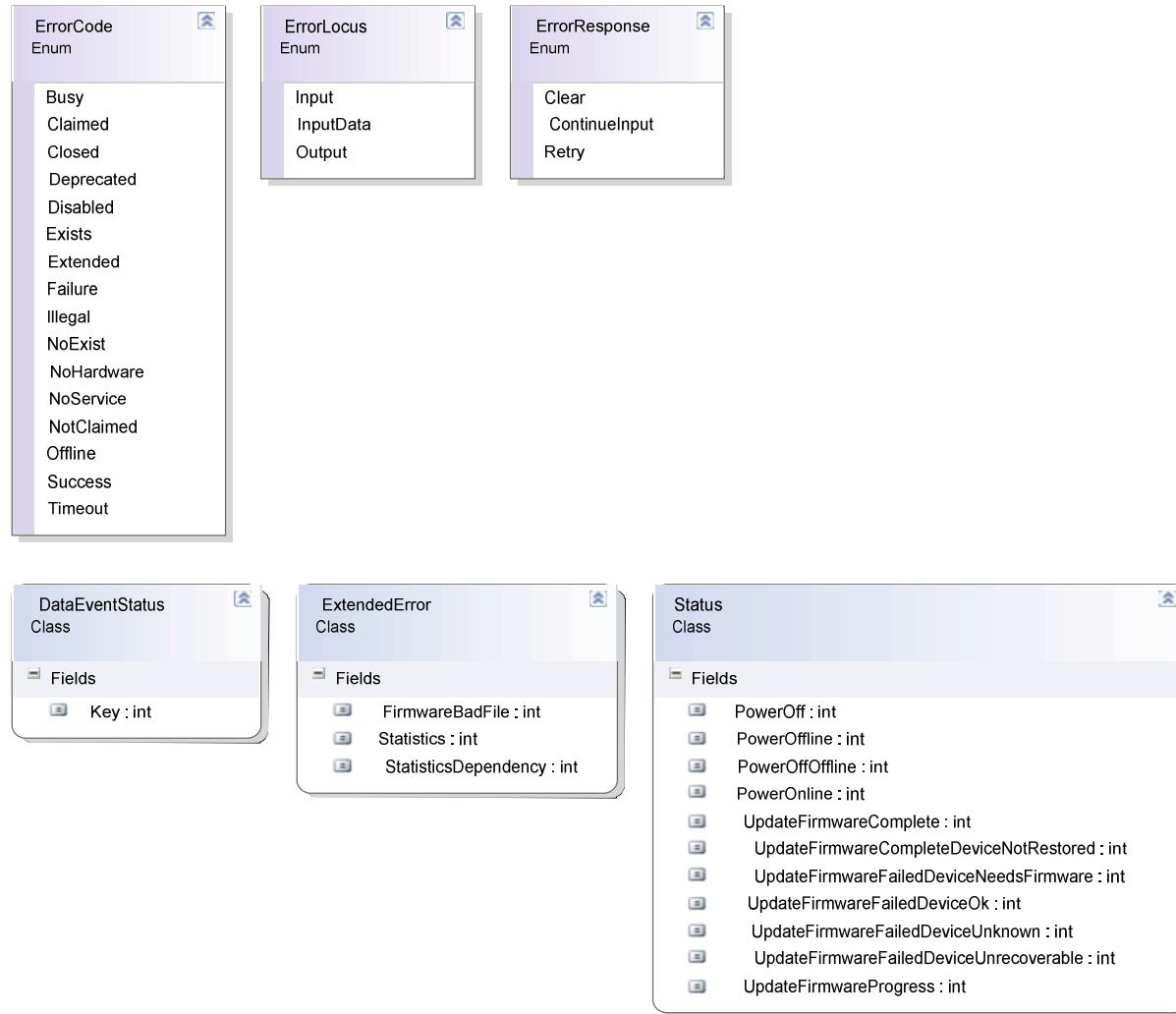
Bump Bar



Bump Bar Events



Bump Bar Events



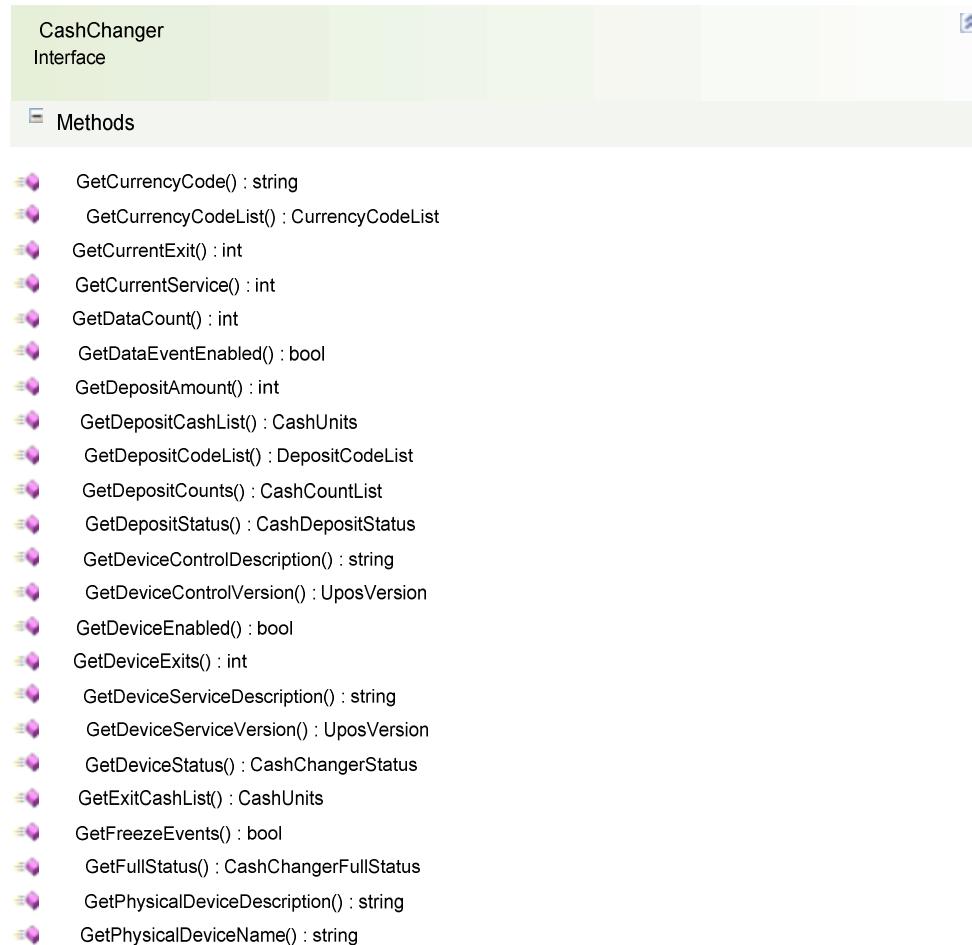
Cash Changer

CashChanger
Interface

The screenshot shows a software interface for the "CashChanger Interface". The interface has a header bar with a file menu icon. Below the header, there is a section titled "Methods" which lists various methods with their descriptions. The methods listed include:

- AdjustCashCounts(CashCountList CashCounts) : void
- BeginDeposit() : void
- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- ClearInput() : void
- Close(string EndpointAddress) : void
 - CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
- DirectIO(int Command, int Data, object Obj) : DirectIOData
- DispenseCash(CashCountList CashCounts) : void
- DispenseChange(int Amount) : void
- EndDeposit(CashDepositAction Success) : void
- FixDeposit() : void
- GetAsyncMode() : bool
- GetAsyncResultCode() : int
- GetAsyncResultCodeExtended() : int
- GetCapCompareFirmwareVersion() : bool
- GetCapDeposit() : bool
- GetCapDepositDataEvent() : bool
- GetCapDiscrepancy() : bool
- GetCapEmptySensor() : bool
- GetCapFullSensor() : bool
- GetCapJamSensor() : bool
- GetCapNearEmptySensor() : bool
- GetCapNearFullSensor() : bool
- GetCapPauseDeposit() : bool
- GetCapPowerReporting() : PowerReporting
- GetCapRealTimeData() : bool
- GetCapRepayDeposit() : bool
- GetCapStatisticsReporting() : bool
- GetCapUpdateFirmware() : bool
- GetCapUpdateStatistics() : bool
- GetCheckHealthText() : string
- GetClaimed() : bool
- GetCurrencyCashList() : CashUnits

Cash Changer

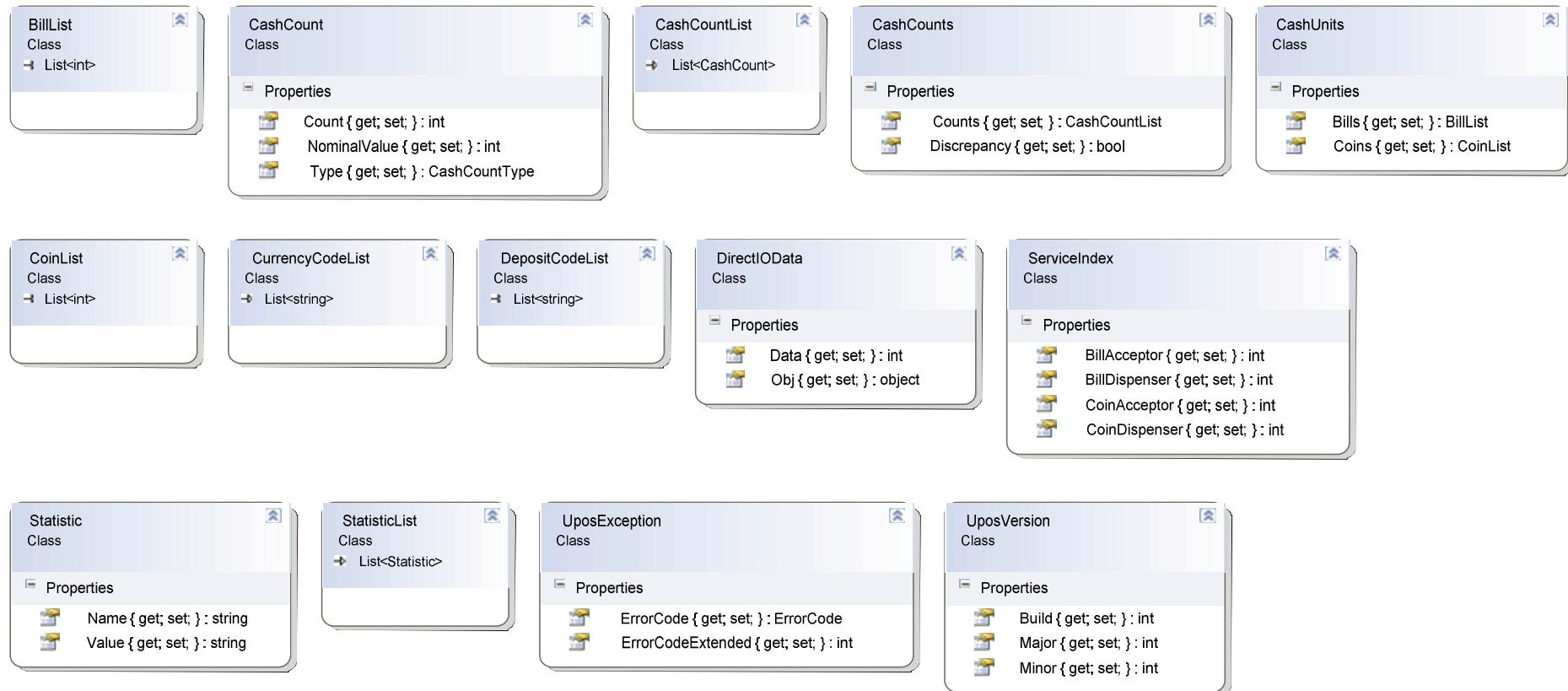


Cash Changer

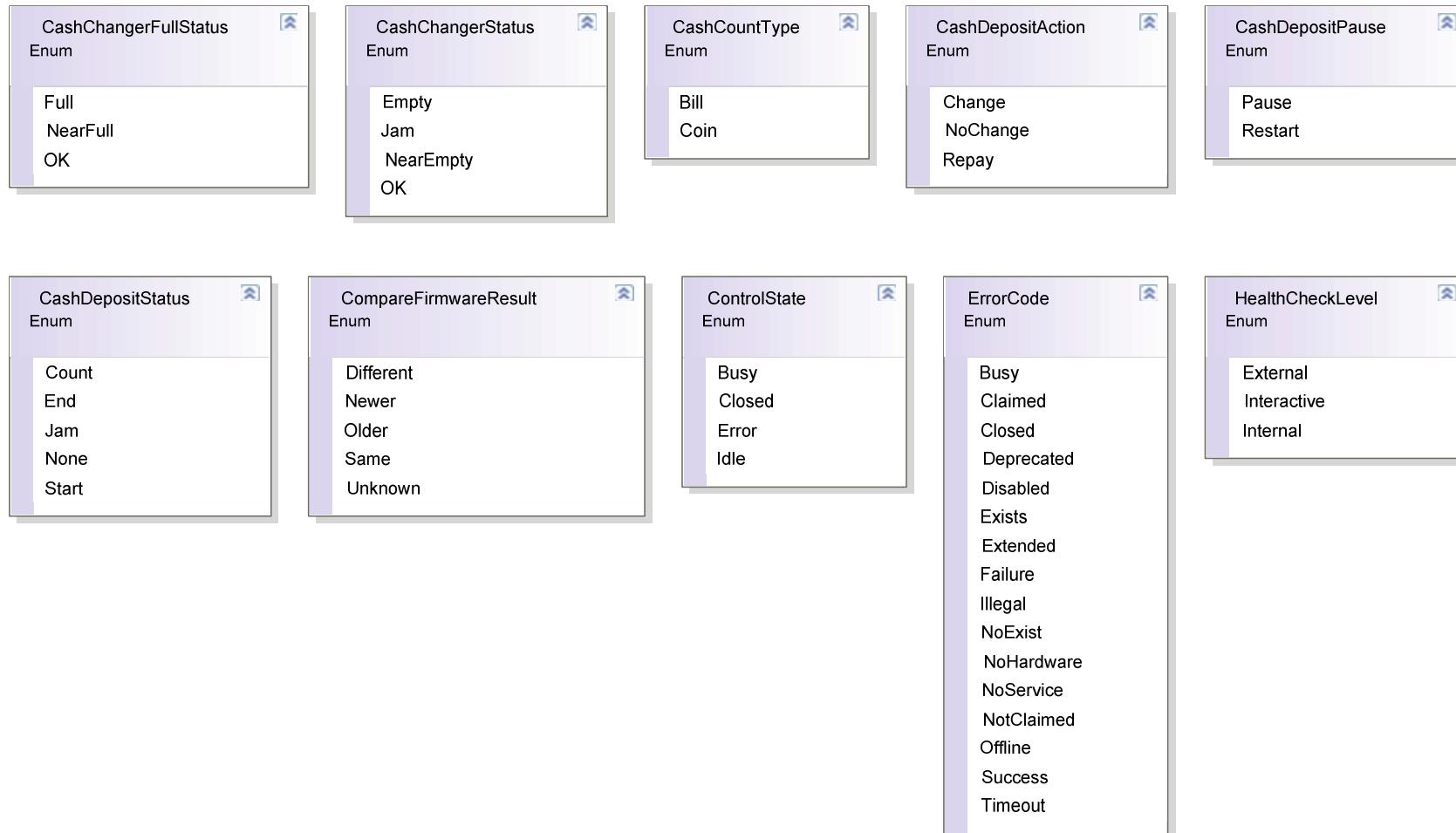
The screenshot shows a software interface with a light green header bar containing the text "CashChanger Interface". Below this is a white main area with a dark grey horizontal bar at the top labeled "Methods". Underneath this bar is a list of 23 methods, each preceded by a small purple icon. The methods are:

- GetPowerNotify() : PowerNotification
- GetPowerState() : PowerState
- GetRealTimeDataEnabled() : bool
- GetServiceCount() : int
- GetServiceIndex() : ServiceIndex
- GetState() : ControlState
- Open(string EndpointAddress) : void
- PauseDeposit(CashDepositPause Control) : void
- ReadCashCounts() : CashCounts
- Release() : void
- ResetStatistics(StatisticList StatisticsBuffer) : void
- RetrieveStatistics(StatisticList StatisticsBuffer) : string
- SetAsyncMode(bool AsyncMode) : void
- SetCurrencyCode(string CurrencyCode) : void
- SetCurrentExit(int CurrentExit) : void
- SetCurrentService(int CurrentService) : void
- SetDataEventEnabled(bool DataEventEnabled) : void
- SetDeviceEnabled(bool DeviceEnabled) : void
- SetFreezeEvents(bool FreezeEvents) : void
- SetPowerNotify(PowerNotification PowerNotify) : void
- SetRealTimeDataEnabled(bool RealTimeDataEnabled) : void
- UpdateFirmware(string FirmwareFileName) : void
- UpdateStatistics(StatisticList StatisticsBuffer) : void

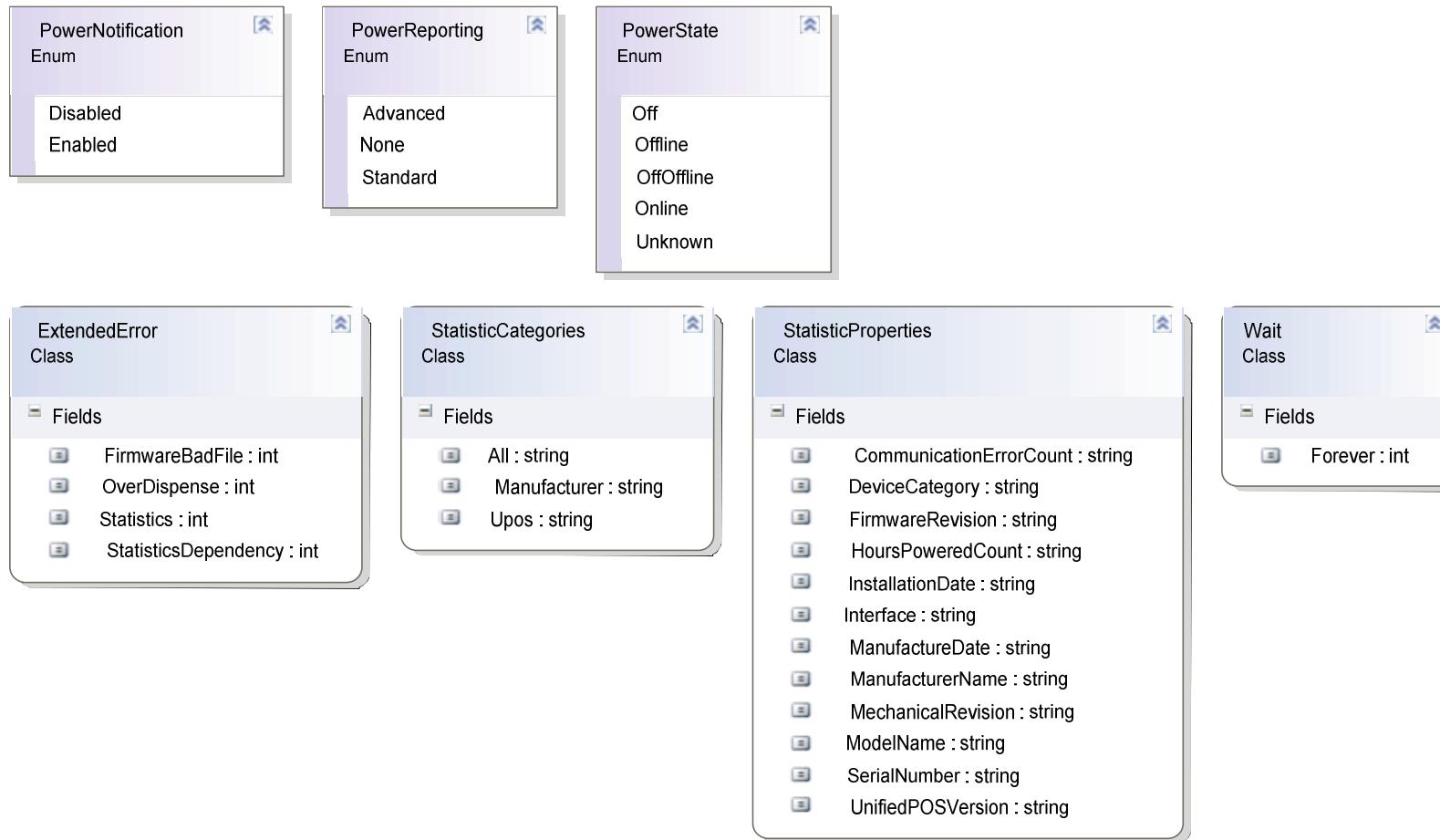
Cash Changer



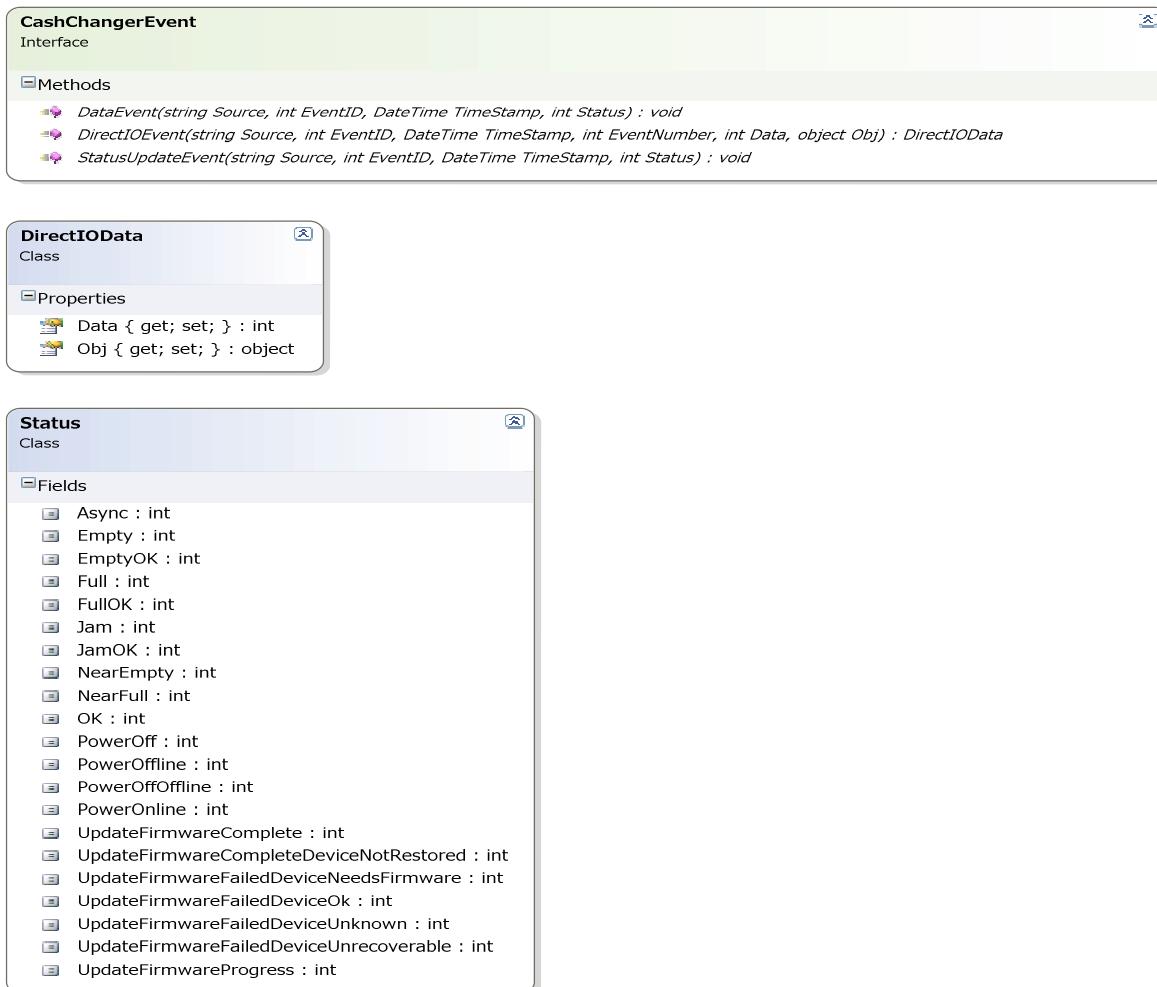
Cash Changer



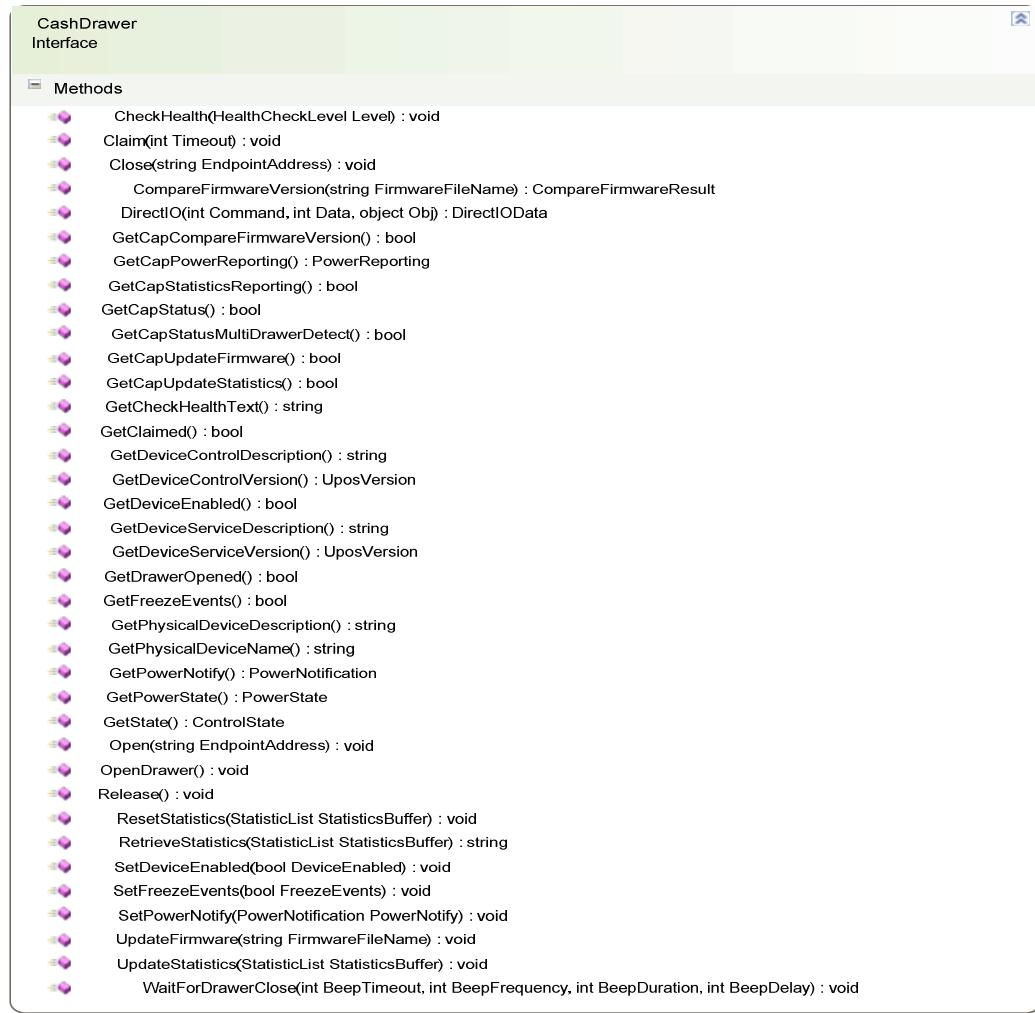
Cash Changer



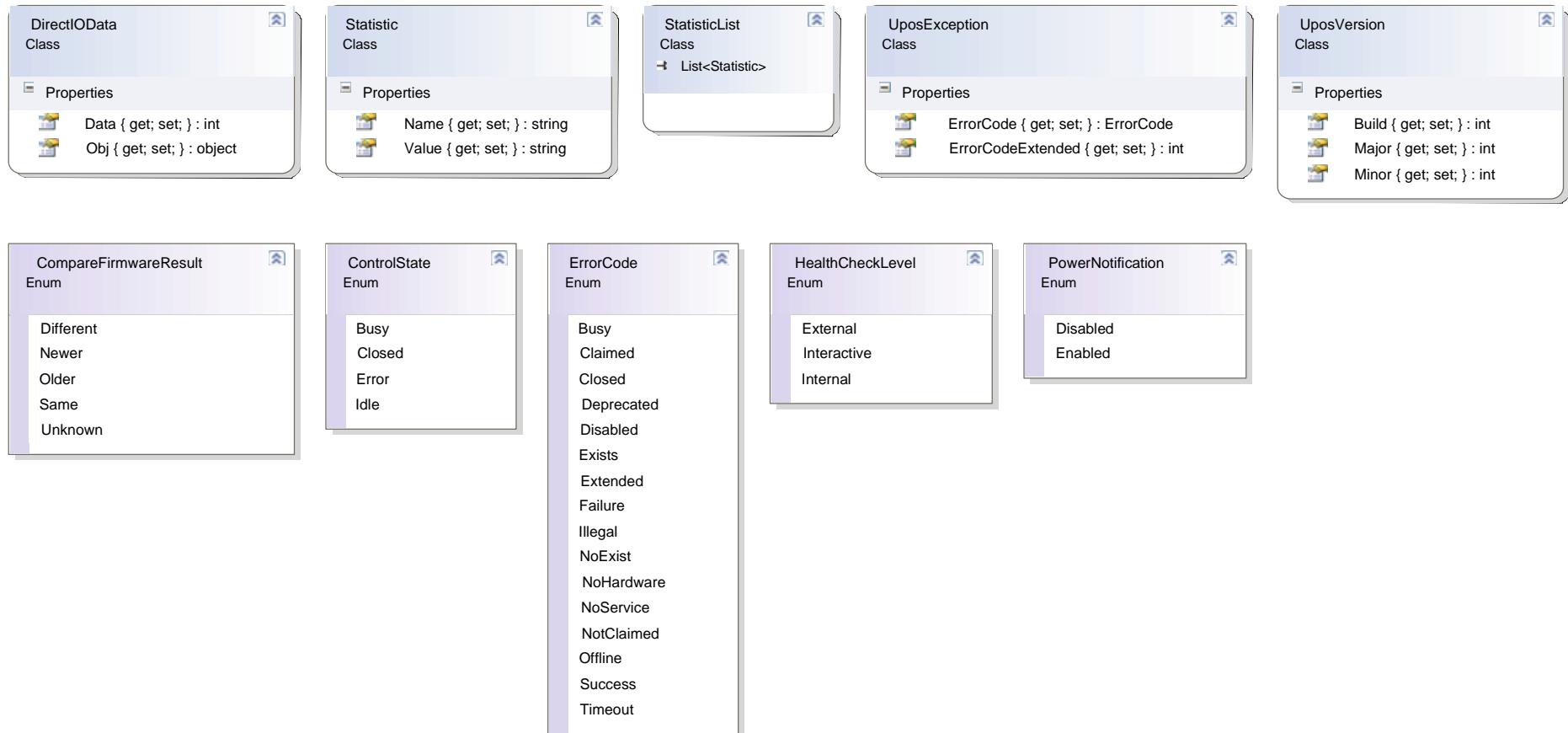
Cash Changer Events



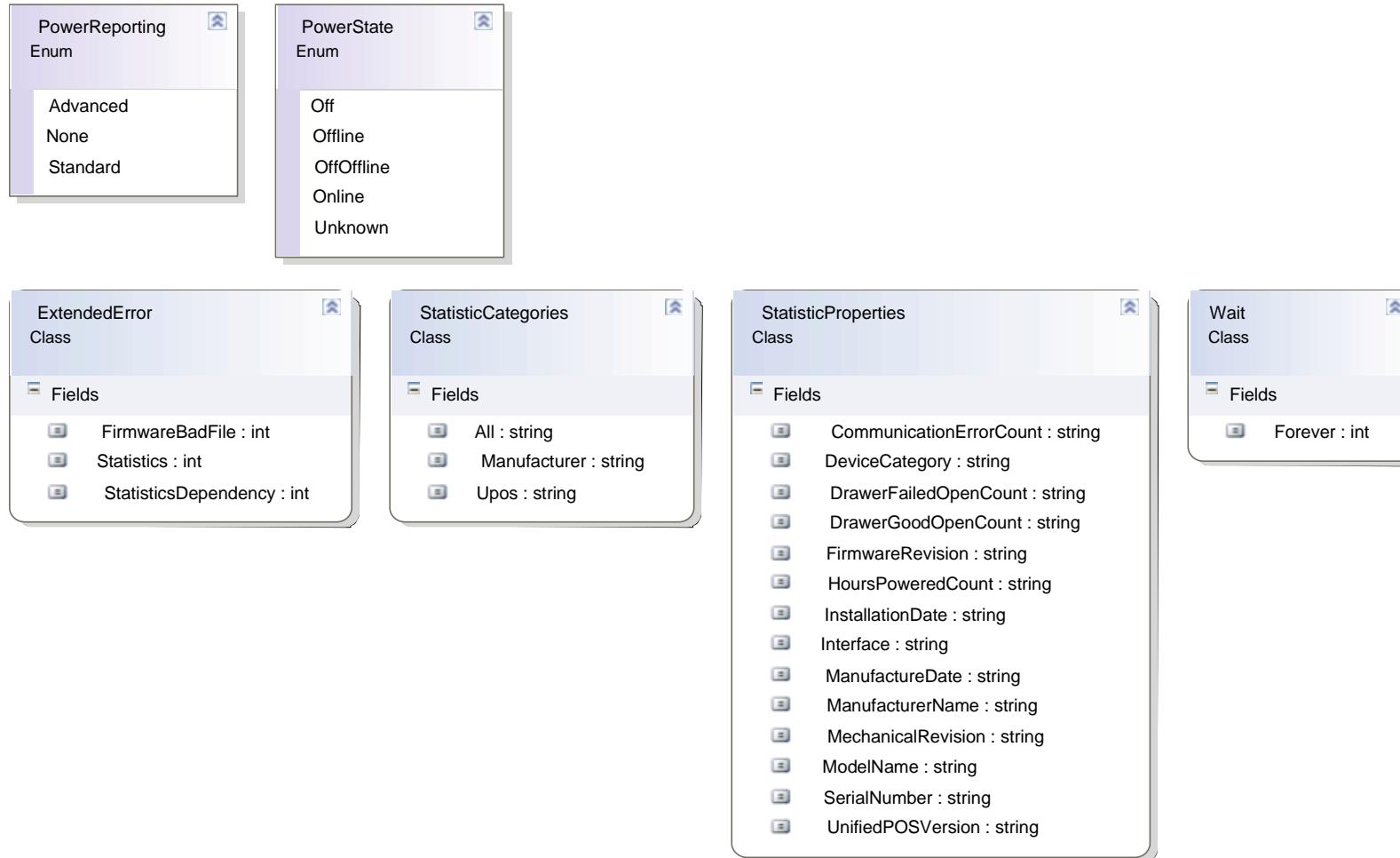
Cash Drawer



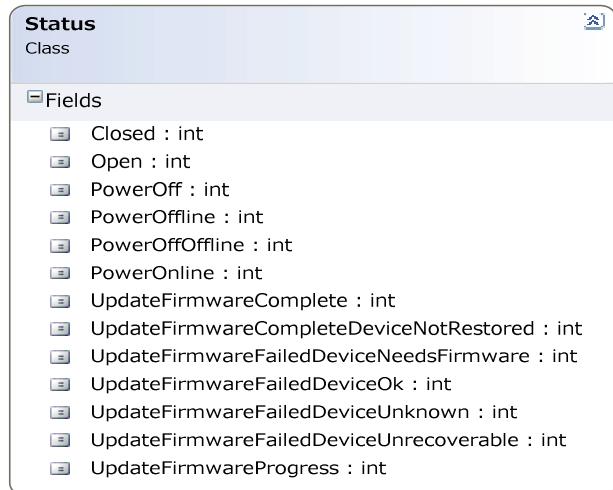
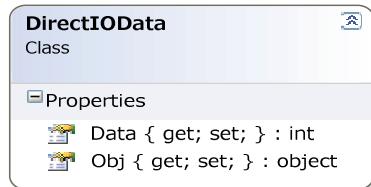
Cash Drawer



Cash Drawer



Cash Drawer Events



CAT (Credit Authorization Terminal)

The screenshot shows a software interface with a light green header bar containing the title 'CAT' and a small icon. Below the header is a sidebar with the title 'CAT Interface' and a 'Methods' section. The 'Methods' section is expanded, showing a list of approximately 40 methods, each preceded by a small purple icon. The methods are listed in two columns:

Method Name	Description
AccessDailyLog(int SequenceNumber, CATLogs Type, int Timeout) : void	
AuthorizeCompletion(int SequenceNumber, decimal Amount, decimal TaxOthers, int Timeout) : void	
AuthorizePreSales(int SequenceNumber, decimal Amount, decimal TaxOthers, int Timeout) : void	
AuthorizeRefund(int SequenceNumber, decimal Amount, decimal TaxOthers, int Timeout) : void	
AuthorizeSales(int SequenceNumber, decimal Amount, decimal TaxOthers, int Timeout) : void	
AuthorizeVoid(int SequenceNumber, decimal Amount, decimal TaxOthers, int Timeout) : void	
AuthorizeVoidPreSales(int SequenceNumber, decimal Amount, decimal TaxOthers, int Timeout) : void	
CashDeposit(int SequenceNumber, decimal Amount, int Timeout) : void	
CheckCard(int SequenceNumber, int Timeout) : void	
CheckHealth(HealthCheckLevel Level) : void	
Claim(int Timeout) : void	
ClearOutput() : void	
Close(string EndpointAddress) : void	
CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult	
DirectIO(int Command, int Data, object Obj) : DirectIOData	
GetAccountNumber() : string	
GetAdditionalSecurityInformation() : string	
GetApprovalCode() : string	
GetAsyncMode() : bool	
GetBalance() : decimal	
GetCapAdditionalSecurityInformation() : bool	
GetCapAuthorizeCompletion() : bool	
GetCapAuthorizePreSales() : bool	
GetCapAuthorizeRefund() : bool	
GetCapAuthorizeVoid() : bool	
GetCapAuthorizeVoidPreSales() : bool	
GetCapCashDeposit() : bool	
GetCapCenterResultCode() : bool	
GetCapCheckCard() : bool	
GetCapCompareFirmwareVersion() : bool	
GetCapDailyLog() : CATLogs	
GetCapInstallments() : bool	
GetCapLockTerminal() : bool	
GetCapLogStatus() : bool	

CAT
Interface

Methods

- GetCapPaymentDetail() : bool
- GetCapPowerReporting() : PowerReporting
- GetCapStatisticsReporting() : bool
- GetCapTaxOthers() : bool
- GetCapTrainingMode() : bool
- GetCapTransactionNumber() : bool
- GetCapUnlockTerminal() : bool
- GetCapUpdateFirmware() : bool
- GetCapUpdateStatistics() : bool
- GetCardCompanyID() : string
- GetCenterResultCode() : string
- GetCheckHealthText() : string
- GetClaimed() : bool
- GetDailyLog() : string
- GetDeviceControlDescription() : string
- GetDeviceControlVersion() : UposVersion
- GetDeviceEnabled() : bool
- GetDeviceServiceDescription() : string
- GetDeviceServiceVersion() : UposVersion
- GetFreezeEvents() : bool
- GetLogStatus() : DealingLogStatus
- GetOutputID() : int
- GetPaymentCondition() : PaymentCondition
- GetPaymentDetail() : string
- GetPaymentMedia() : PaymentMedia
- GetPhysicalDeviceDescription() : string
- GetPhysicalDeviceName() : string

CAT (Credit Authorization Terminal)

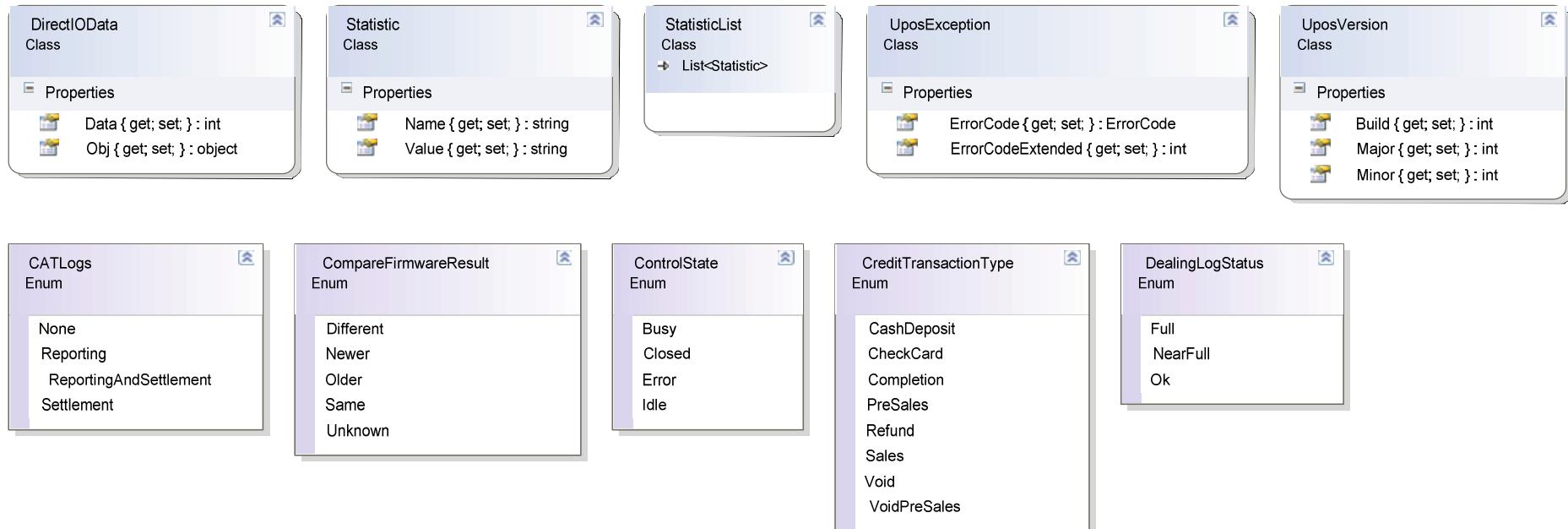
CAT
Interface



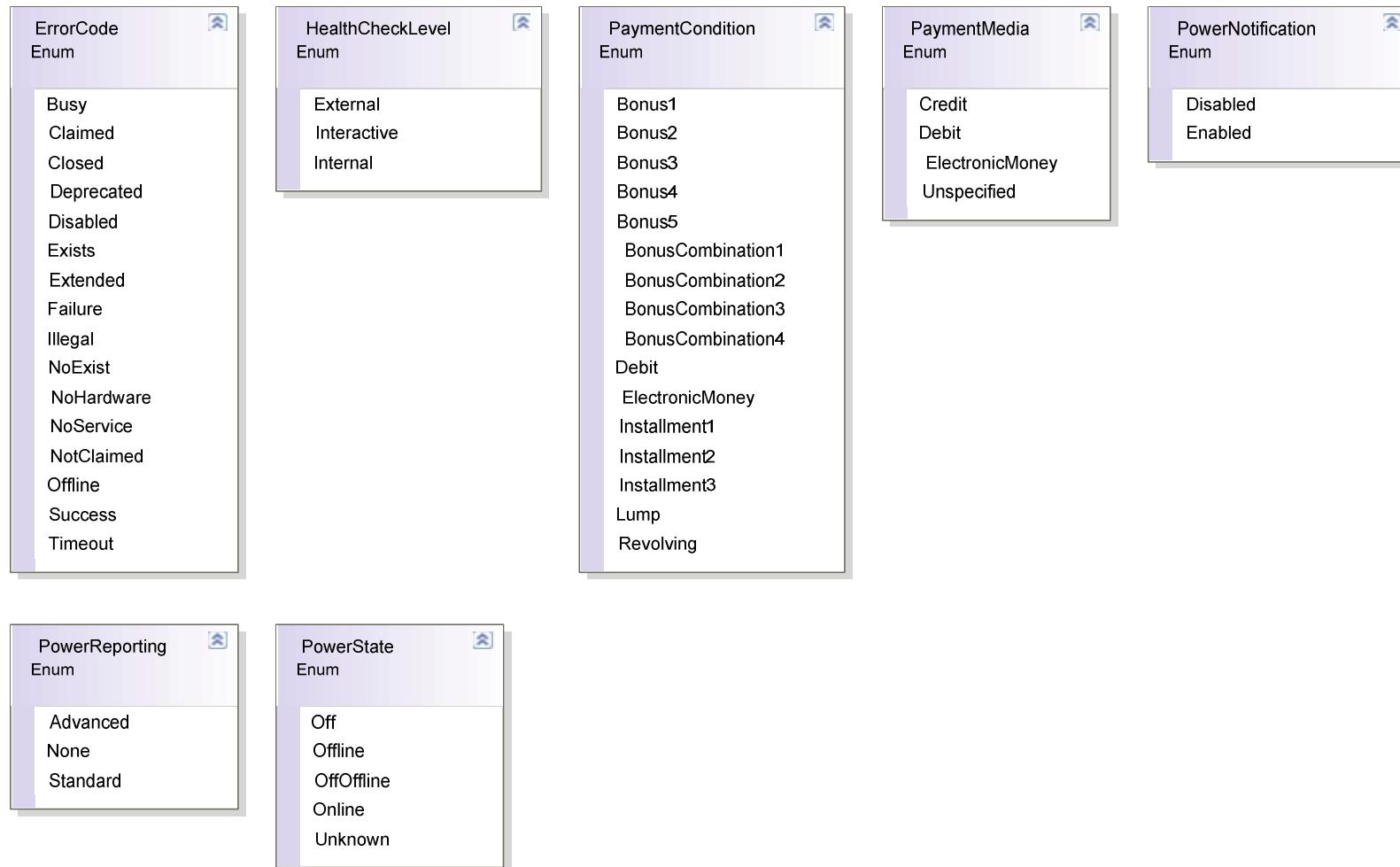
Methods

- ≡ GetPowerNotify() : PowerNotification
- ≡ GetPowerState() : PowerState
- ≡ GetSequenceNumber() : int
- ≡ GetSettledAmount() : decimal
- ≡ GetSlipNumber() : string
- ≡ GetState() : ControlState
- ≡ GetTrainingMode() : bool
- ≡ GetTransactionNumber() : string
- ≡ GetTransactionType() : CreditTransactionType
- ≡ LockTerminal() : void
- ≡ Open(string EndpointAddress) : void
- ≡ Release() : void
- ≡ ResetStatistics(StatisticList StatisticsBuffer) : void
- ≡ RetrieveStatistics(StatisticList StatisticsBuffer) : string
- ≡ SetAdditionalSecurityInformation(string AdditionalSecurityInformation) : void
- ≡ SetAsyncMode(bool AsyncMode) : void
- ≡ SetDeviceEnabled(bool DeviceEnabled) : void
- ≡ SetFreezeEvents(bool FreezeEvents) : void
- ≡ SetPaymentMedia(PaymentMedia PaymentMedia) : void
- ≡ SetPowerNotify(PowerNotification PowerNotify) : void
- ≡ SetTrainingMode(bool TrainingMode) : void
- ≡ UnlockTerminal() : void
- ≡ UpdateFirmware(string FirmwareFileName) : void
- ≡ UpdateStatistics(StatisticList StatisticsBuffer) : void

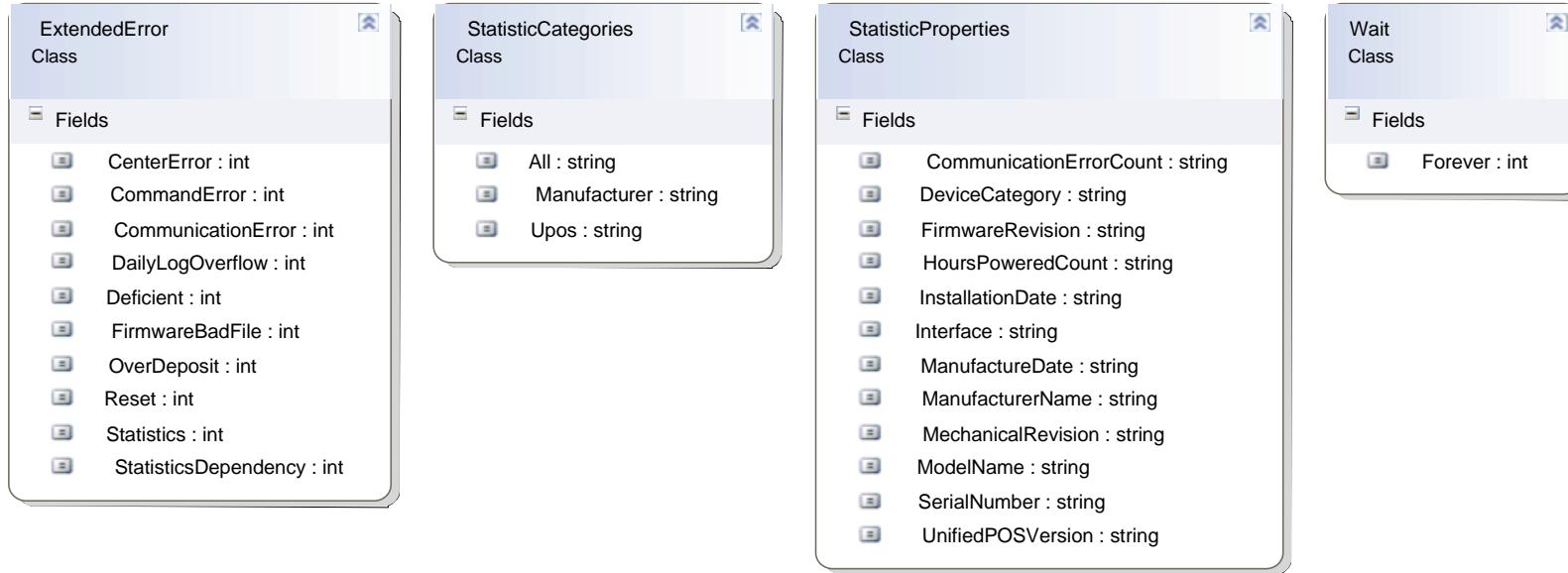
CAT (Credit Authorization Terminal)



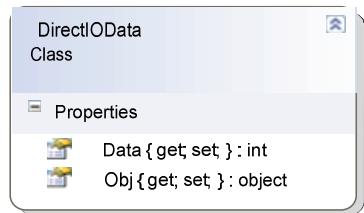
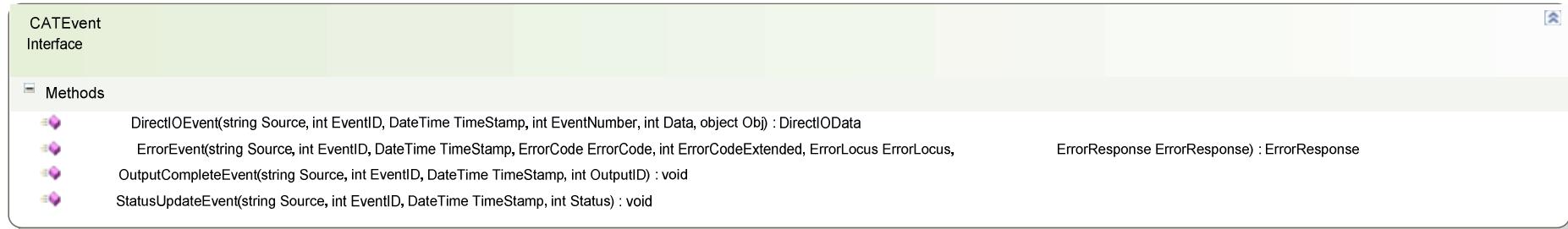
CAT (Credit Authorization Terminal)



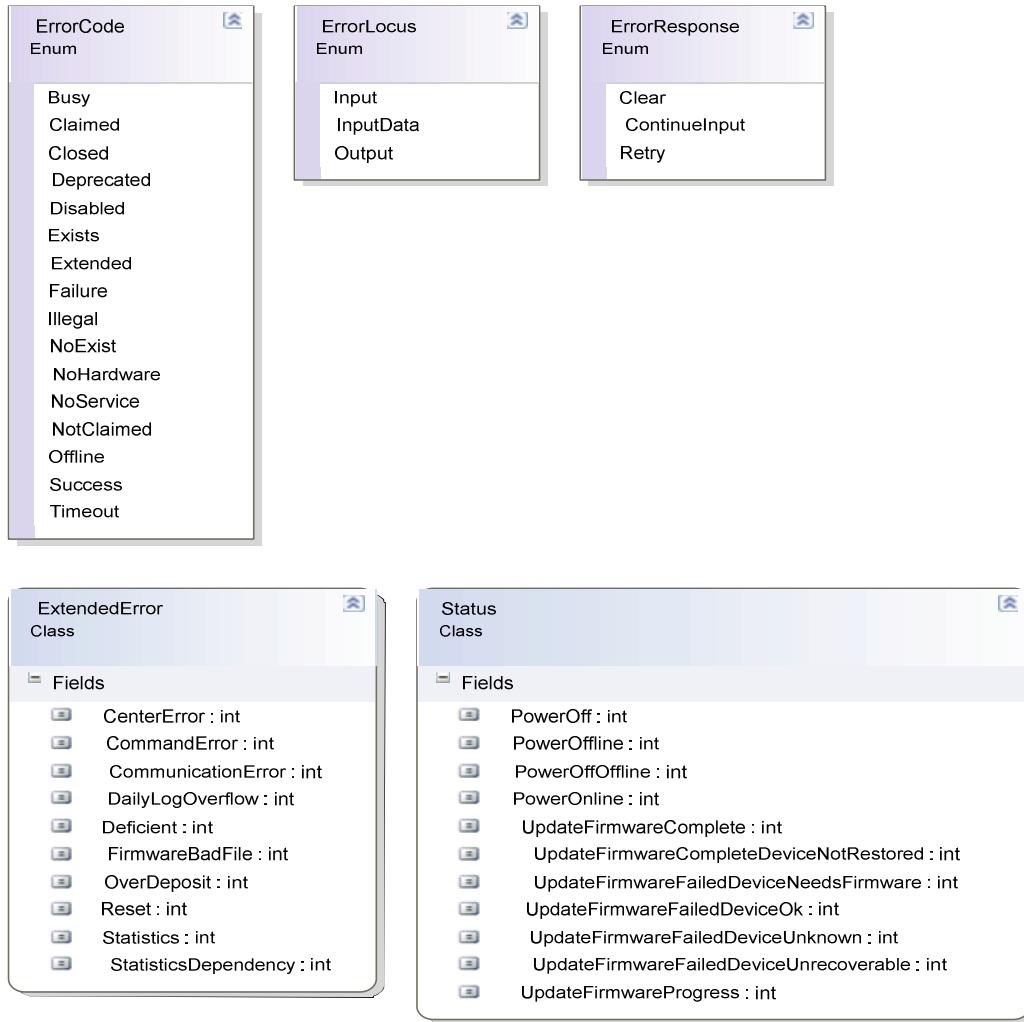
CAT (Credit Authorization Terminal)



CAT (Credit Authorization Terminal) Events



CAT (Credit Authorization Terminal) Events



Check Scanner

CheckScanner
Interface

Methods

- BeginInsertion(int Timeout) : void
- BeginRemoval(int Timeout) : void
- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- ClearImage(CheckImageClear By) : void
- ClearInput() : void
- ClearInputProperties() : void
- Close(string EndpointAddress) : void
 - CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
- DefineCropArea(int CropAreaID, int X, int CX, int CY) : void
- DirectIO(int Command, int Data, object Obj) : DirectIOData
- EndInsertion() : void
- EndRemoval() : void
- GetAutoDisable() : bool
- GetCapAutoContrast() : bool
- GetCapAutoGenerateFileID() : bool
- GetCapAutoGenerateImageTagData() : bool
- GetCapAutoSize() : bool
- GetCapColor() : int
 - GetCapCompareFirmwareVersion() : bool
 - GetCapConcurrentMICR() : bool
 - GetCapContrast() : bool
 - GetCapDefineCropArea() : bool
 - GetCapImageFormat() : int
 - GetCapImageTagData() : bool
 - GetCapMICRDevice() : bool
 - GetCapPowerReporting() : PowerReporting
 - GetCapStatisticsReporting() : bool
 - GetCapStoreImageFiles() : bool
 - GetCapUpdateFirmware() : bool

Check Scanner



The screenshot shows a software interface with a light green header bar containing the text "CheckScanner Interface". Below this is a white area with a tree view. The tree view has a single expanded node labeled "Methods", which contains a list of method names. Each method name is preceded by a small purple icon.

- ≡ GetCapUpdateStatistics() : bool
- ≡ GetCapValidationDevice() : bool
- ≡ GetCheckHealthText() : string
- ≡ GetClaimed() : bool
- ≡ GetColor() : int
- ≡ GetConcurrentMICR() : bool
- ≡ GetContrast() : int
- ≡ GetCropAreaCount() : int
- ≡ GetDataCount() : int
- ≡ GetDataEventEnabled() : bool
- ≡ GetDeviceControlDescription() : string
- ≡ GetDeviceControlVersion() : UposVersion
- ≡ GetDeviceEnabled() : bool
- ≡ GetDeviceServiceDescription() : string
- ≡ GetDeviceServiceVersion() : UposVersion
- ≡ GetDocumentHeight() : int
- ≡ GetDocumentWidth() : int
- ≡ GetFileID() : string
- ≡ GetFileIndex() : int
- ≡ GetFreezeEvents() : bool
- ≡ GetImageData() : byte[]
- ≡ GetImageFormat() : int
- ≡ GetImageMemoryStatus() : ImageMemoryStatus

Check Scanner



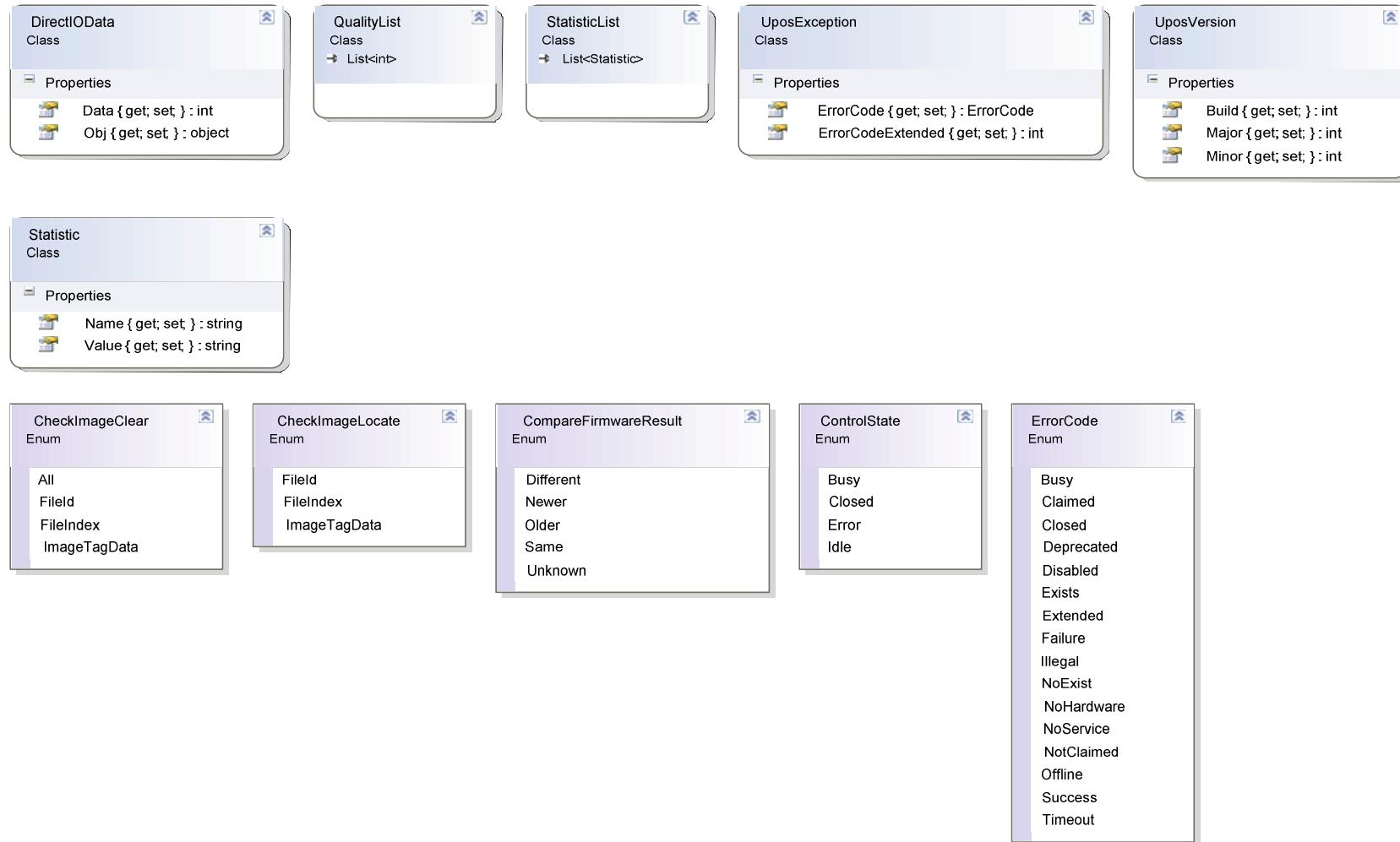
- ≡ GetImageTagData() : string
- ≡ GetMapMode() : MapMode
- ≡ GetMaxCropAreas() : int
- ≡ GetPhysicalDeviceDescription() : string
- ≡ GetPhysicalDeviceName() : string
- ≡ GetPowerNotify() : PowerNotification
- ≡ GetPowerState() : PowerState
- ≡ GetQuality() : int
- ≡ GetQualityList() : QualityList
- ≡ GetRemainingImagesEstimate() : int
- ≡ GetState() : ControlState
- ≡ Open(string EndpointAddress) : void
- ≡ Release() : void
- ≡ ResetStatistics(StatisticList StatisticsBuffer) : void
- ≡ RetrieveImage(int CropAreaID) : void
- ≡ RetrieveMemory(CheckImageLocate By) : void
- ≡ RetrieveStatistics(StatisticList StatisticsBuffer) : string
- ≡ SetAutoDisable(bool AutoDisable) : void

Check Scanner

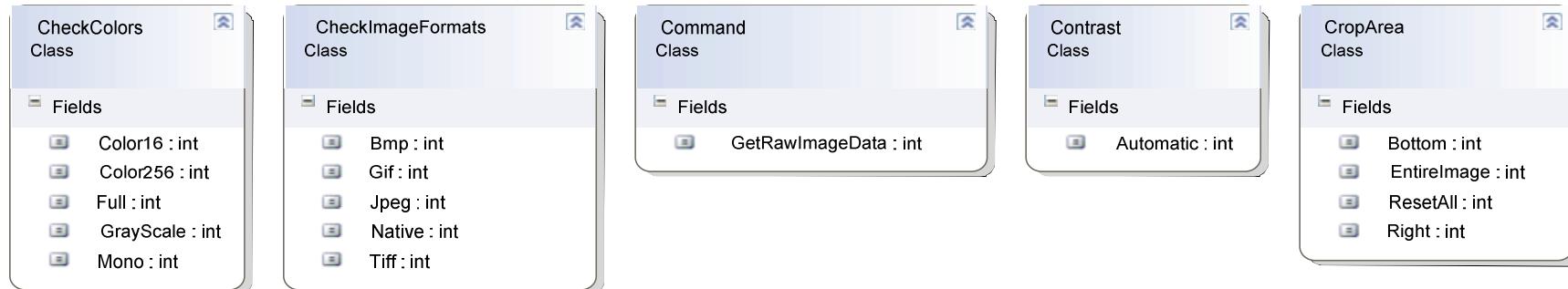
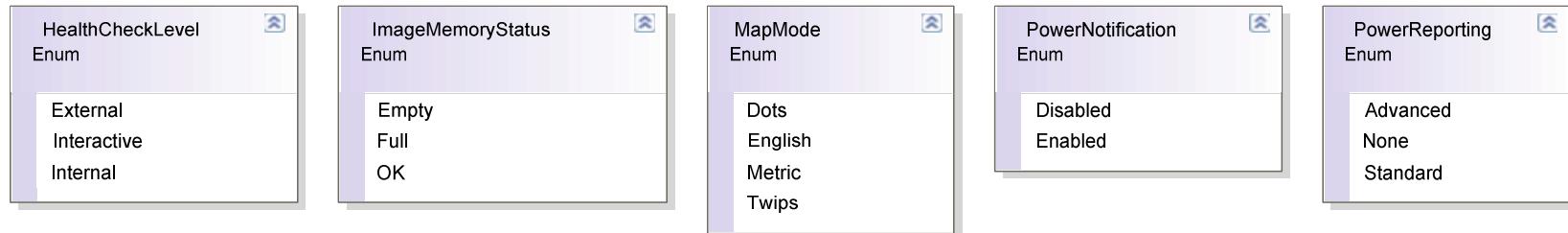


- SetColor(int Color) : void
- SetConcurrentMICR(bool ConcurrentMICR) : void
- SetContrast(int Contrast) : void
- SetDataEventEnabled(bool DataEventEnabled) : void
- SetDeviceEnabled(bool DeviceEnabled) : void
- SetDocumentHeight(int DocumentHeight) : void
- SetDocumentWidth(int DocumentWidth) : void
- SetFileID(string FileID) : void
- SetFileIndex(int FileIndex) : void
- SetFreezeEvents(bool FreezeEvents) : void
- SetImageFormat(int ImageFormat) : void
- SetImageTagData(string ImageTagData) : void
- SetMapMode(MapMode MapMode) : void
- SetPowerNotify(PowerNotification PowerNotify) : void
- SetQuality(int Quality) : void
- StoreImage(int CropAreaID) : void
- UpdateFirmware(string FirmwareFileName) : void
- UpdateStatistics(StatisticList StatisticsBuffer) : void

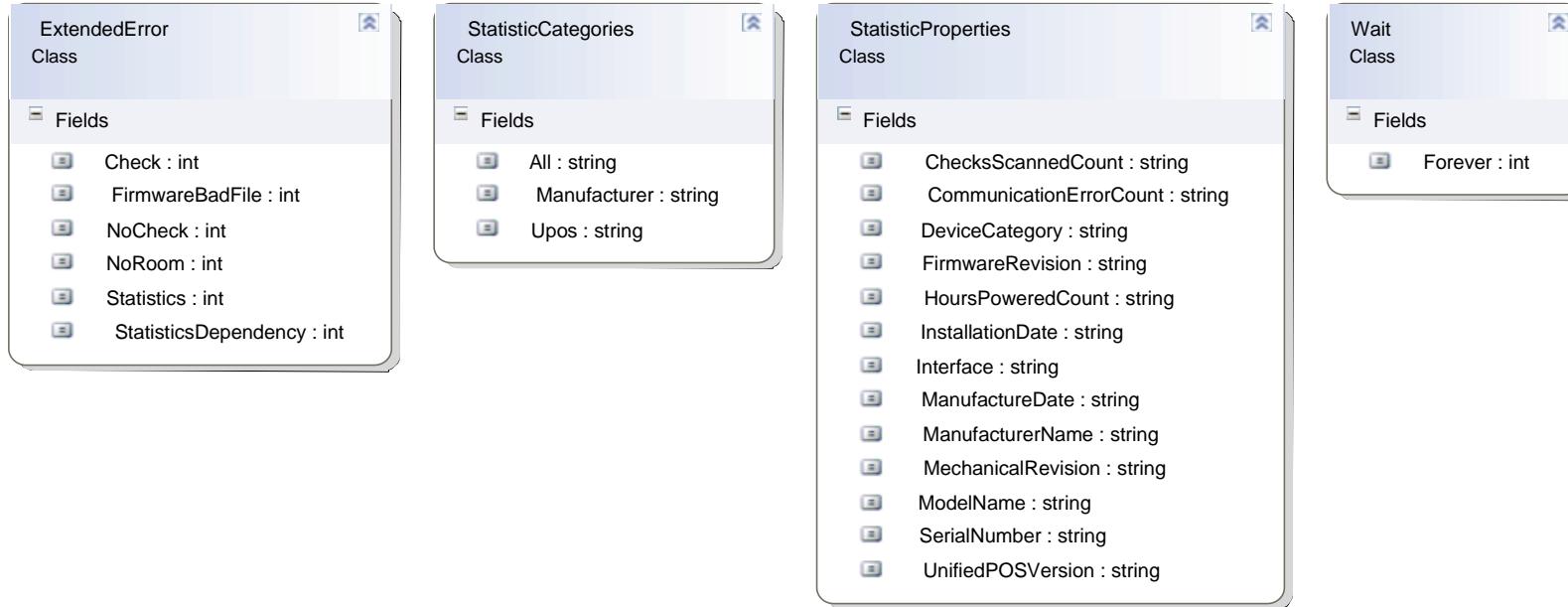
Check Scanner



Check Scanner



Check Scanner



Check Scanner Events

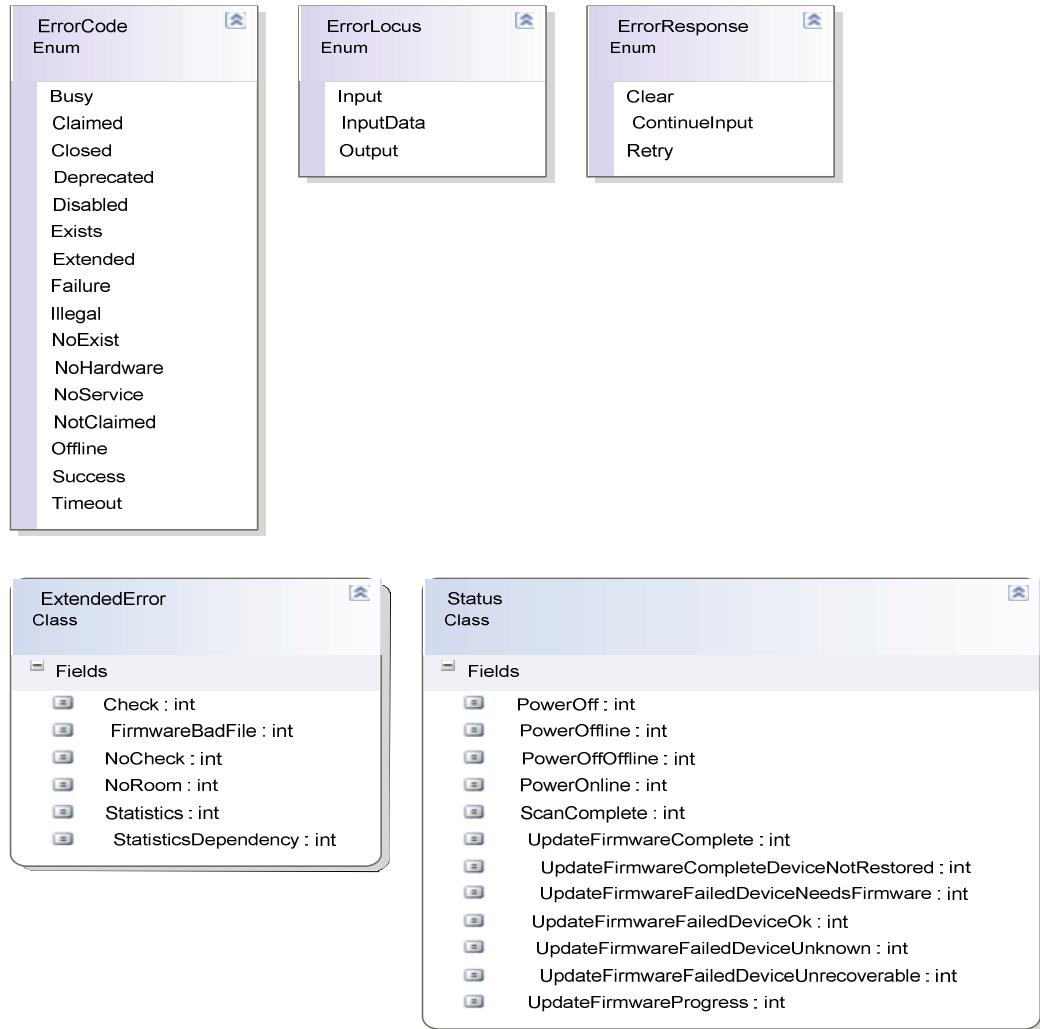
The diagram illustrates the UML class structure for the Check Scanner Events. It consists of two main components:

- CheckScannerEvent Interface:** A light green box representing an interface. It contains:
 - A small icon in the top right corner.
 - The text "CheckScannerEvent" followed by "Interface".
 - A section titled "Methods" with four entries, each preceded by a small purple icon:
 - DataEvent(string Source, int EventID, DateTime TimeStamp, int Status) : void
 - DirectIOEvent(string Source, int EventID, DateTime TimeStamp, int EventNumber, int Data, object Obj) : DirectIOData
 - ErrorEvent(string Source, int EventID, DateTime TimeStamp, ErrorCode ErrorCode, int ErrorCodeExtended, ErrorLocus ErrorLocus, ErrorResponse ErrorResponse) : ErrorResponse
 - StatusUpdateEvent(string Source, int EventID, DateTime TimeStamp, int Status) : void
- DirectIOData Class:** A white box representing a class. It contains:
 - A small icon in the top right corner.
 - The text "DirectIOData" followed by "Class".
 - A section titled "Properties" with two entries, each preceded by a small icon:
 - Data { get; set; } : int
 - Obj { get; set; } : object

This diagram provides a detailed view of the DirectIOData class properties. It shows the class name "DirectIOData" and "Class" at the top, followed by a "Properties" section containing two entries:

- Data { get; set; } : int
- Obj { get; set; } : object

Check Scanner Events



Coin Acceptor

CoinAcceptor
Interface

Methods

- AdjustCashCounts(CashCountList CashCounts) : void
- BeginDeposit() : void
- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- ClearInput() : void
- Close(string EndpointAddress) : void
 - CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
- DirectIO(int Command, int Data, object Obj) : DirectIOData
- EndDeposit(EndDepositAction Success) : void
- FixDeposit() : void
- GetCapCompareFirmwareVersion() : bool
- GetCapDiscrepancy() : bool
- GetCapFullSensor() : bool
- GetCapJamSensor() : bool
- GetCapNearFullSensor() : bool
- GetCapPauseDeposit() : bool
- GetCapPowerReporting() : PowerReporting
- GetCapRealTimeData() : bool
- GetCapStatisticsReporting() : bool
- GetCapUpdateFirmware() : bool
- GetCapUpdateStatistics() : bool
- GetCheckHealthText() : string
- GetClaimed() : bool

Coin Acceptor

The diagram shows a UML class named "CoinAcceptor Interface". It has a single section labeled "Methods" which contains the following list of methods:

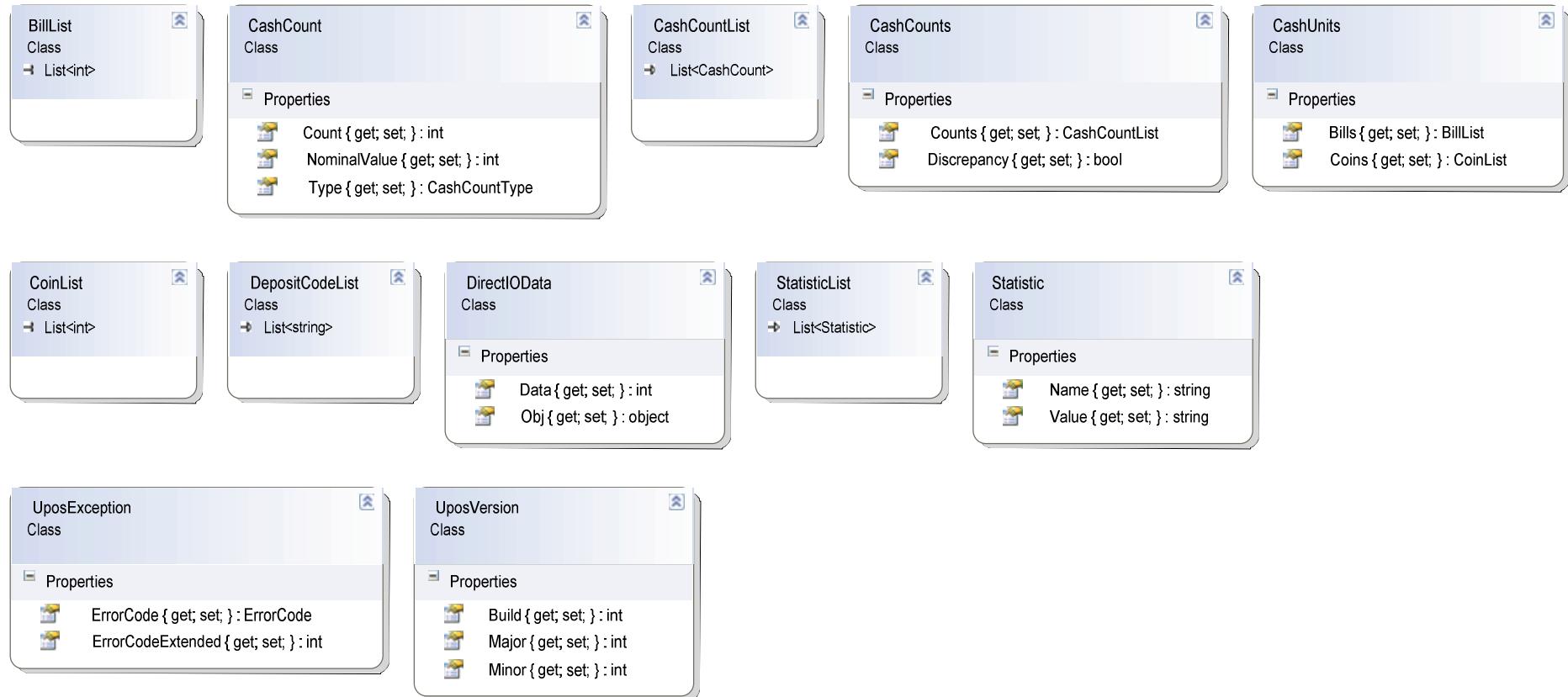
- GetCurrencyCode() : string
- GetDataCount() : int
- GetDataEventEnabled() : bool
- GetDepositAmount() : int
- GetDepositCashList() : CashUnits
- GetDepositCodeList() : DepositCodeList
- GetDepositCounts() : CashCountList
- GetDepositStatus() : DepositStatus
- GetDeviceControlDescription() : string
- GetDeviceControlVersion() : UposVersion
- GetDeviceEnabled() : bool
- GetDeviceServiceDescription() : string
- GetDeviceServiceVersion() : UposVersion
- GetFreezeEvents() : bool
- GetFullStatus() : CashChangerFullStatus
- GetPhysicalDeviceDescription() : string
- GetPhysicalDeviceName() : string
- GetPowerNotify() : PowerNotification
- GetPowerState() : PowerState
- GetRealTimeDataEnabled() : bool
- GetState() : ControlState
- Open(string EndpointAddress) : void

Coin Acceptor

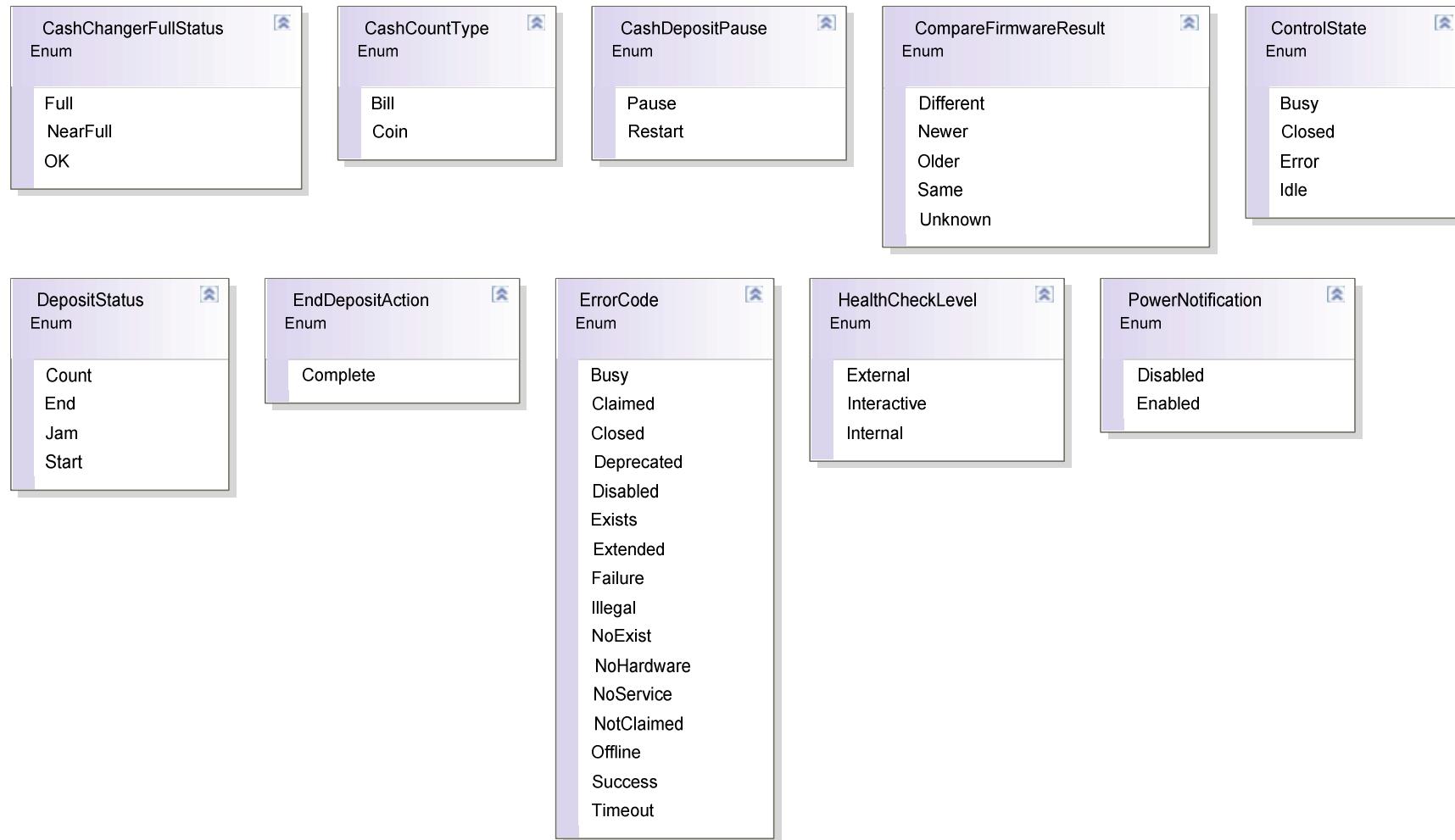


- ⌚ PauseDeposit(CashDepositPause Control) : void
- ⌚ ReadCashCounts() : CashCounts
- ⌚ Release() : void
- ⌚ ResetStatistics(StatisticList StatisticsBuffer) : void
- ⌚ RetrieveStatistics(StatisticList StatisticsBuffer) : string
- ⌚ SetCurrencyCode(string CurrencyCode) : void
- ⌚ SetDataEventEnabled(bool DataEventEnabled) : void
- ⌚ SetDeviceEnabled(bool DeviceEnabled) : void
- ⌚ SetFreezeEvents(bool FreezeEvents) : void
- ⌚ SetPowerNotify(PowerNotification PowerNotify) : void
- ⌚ SetRealTimeDataEnabled(bool RealTimeDataEnabled) : void
- ⌚ UpdateFirmware(string FirmwareFileName) : void
- ⌚ UpdateStatistics(StatisticList StatisticsBuffer) : void

Coin Acceptor

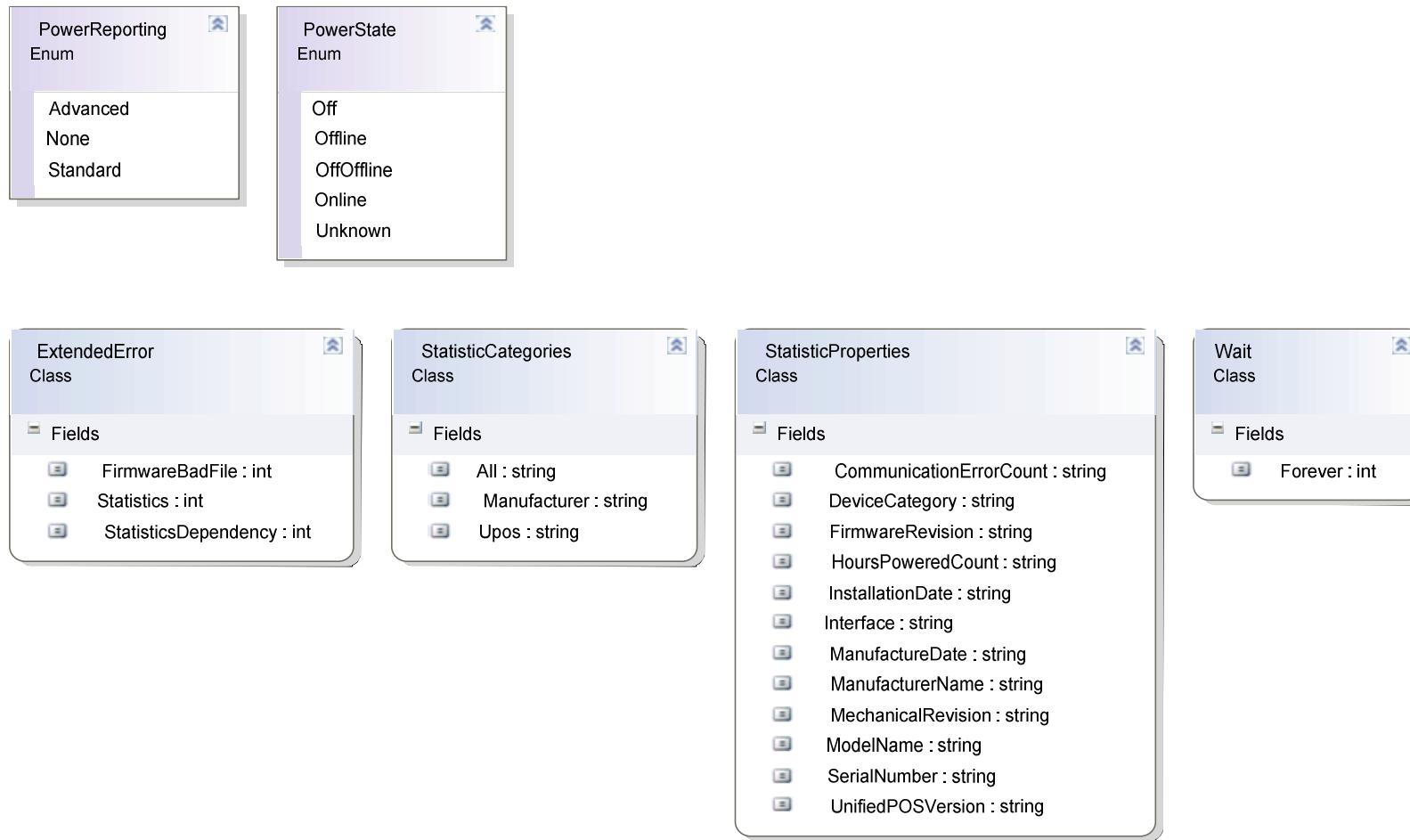


Coin Acceptor

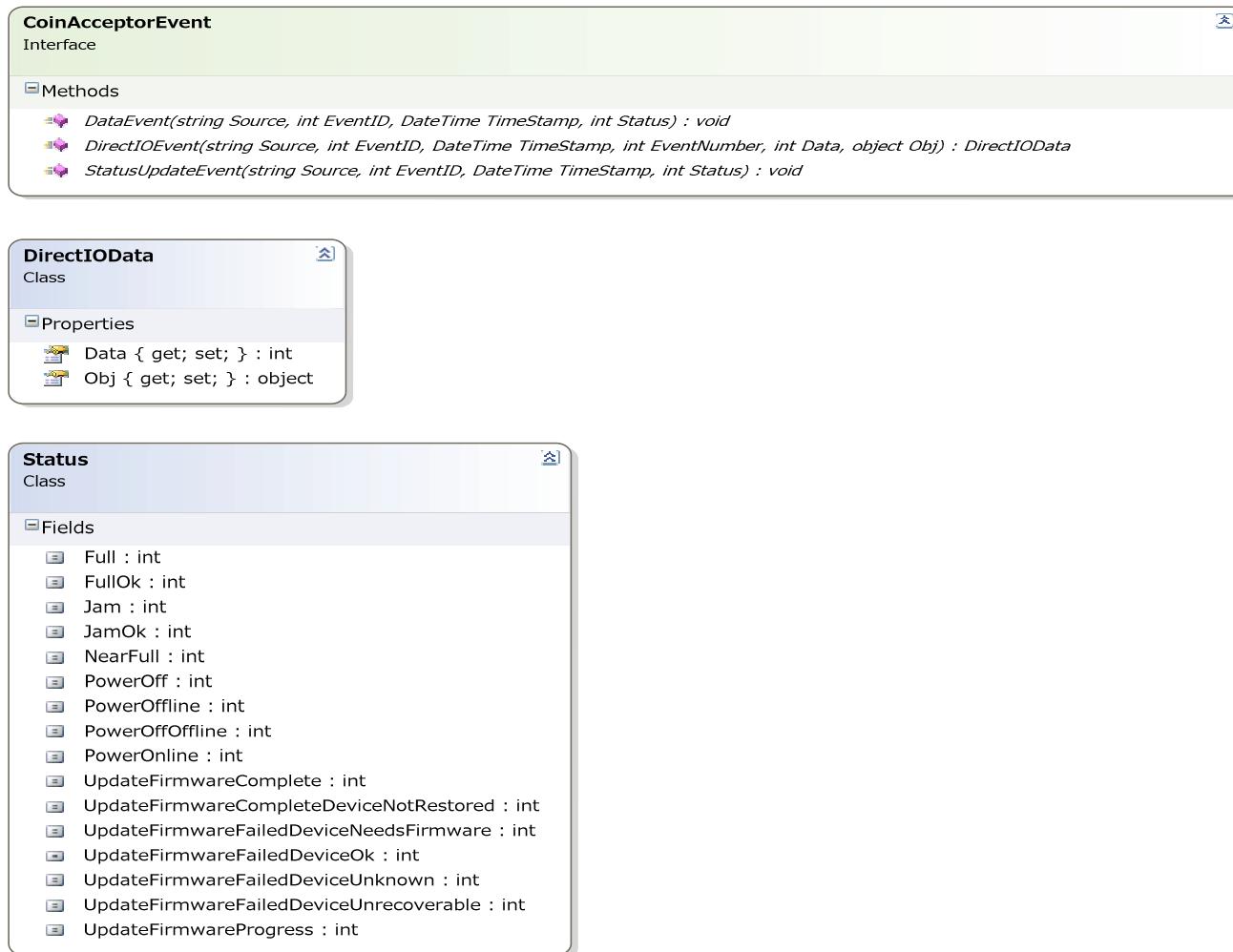


WS-POS 1.1 Technical Specification

Coin Acceptor



Coin Acceptor Events



Coin Dispenser

The diagram shows a UML class diagram for the **CoinDispenser Interface**. The interface is represented by a light green rectangle. Below it, a list of methods is shown under a section titled "Methods". Each method is preceded by a small icon representing its return type.

- AdjustCashCounts(CashCountList CashCounts) : void
- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- Close(string EndpointAddress) : void
- CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
- DirectIO(int Command, int Data, object Obj) : DirectIOData
- DispenseChange(int Amount) : void
- GetCapCompareFirmwareVersion() : bool
- GetCapEmptySensor() : bool
- GetCapJamSensor() : bool
- GetCapNearEmptySensor() : bool
- GetCapPowerReporting() : PowerReporting
- GetCapStatisticsReporting() : bool
- GetCapUpdateFirmware() : bool
- GetCapUpdateStatistics() : bool
- GetCheckHealthText() : string
- GetClaimed() : bool
- GetDeviceControlDescription() : string
- GetDeviceControlVersion() : UposVersion
- GetDeviceEnabled() : bool

Coin Dispenser

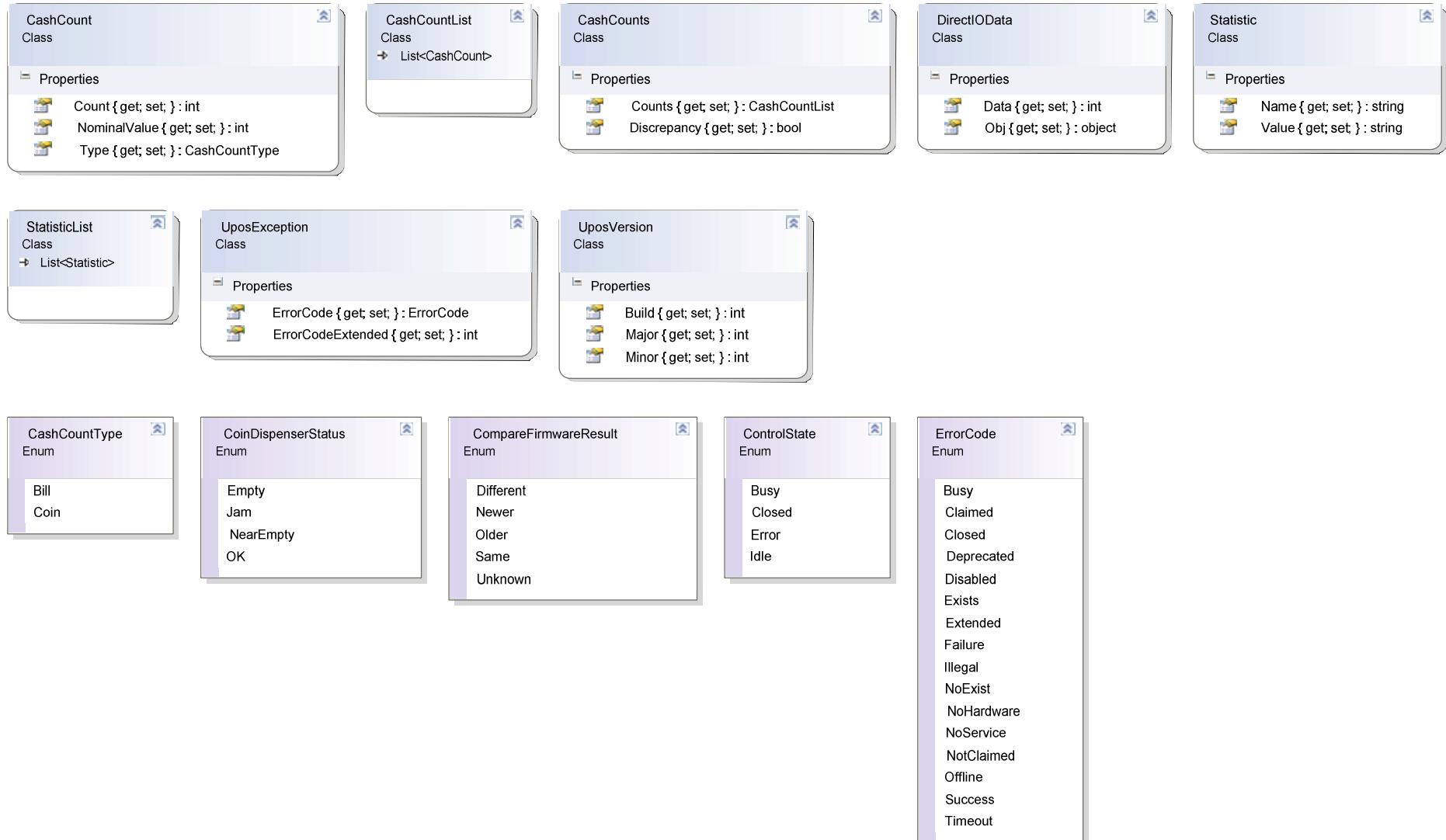


Methods

- GetDeviceServiceDescription() : string
- GetDeviceServiceVersion() : UposVersion
- GetDispenserStatus() : CoinDispenserStatus
- GetFreezeEvents() : bool
- GetPhysicalDeviceDescription() : string
- GetPhysicalDeviceName() : string
- GetPowerNotify() : PowerNotification
- GetPowerState() : PowerState
- GetState() : ControlState
- Open(string EndpointAddress) : void
- ReadCashCounts() : CashCounts
- Release() : void
- ResetStatistics(StatisticList StatisticsBuffer) : void
- RetrieveStatistics(StatisticList StatisticsBuffer) : string
- SetDeviceEnabled(bool DeviceEnabled) : void
- SetFreezeEvents(bool FreezeEvents) : void
- SetPowerNotify(PowerNotification PowerNotify) : void
- UpdateFirmware(string FirmwareFileName) : void
- UpdateStatistics(StatisticList StatisticsBuffer) : void

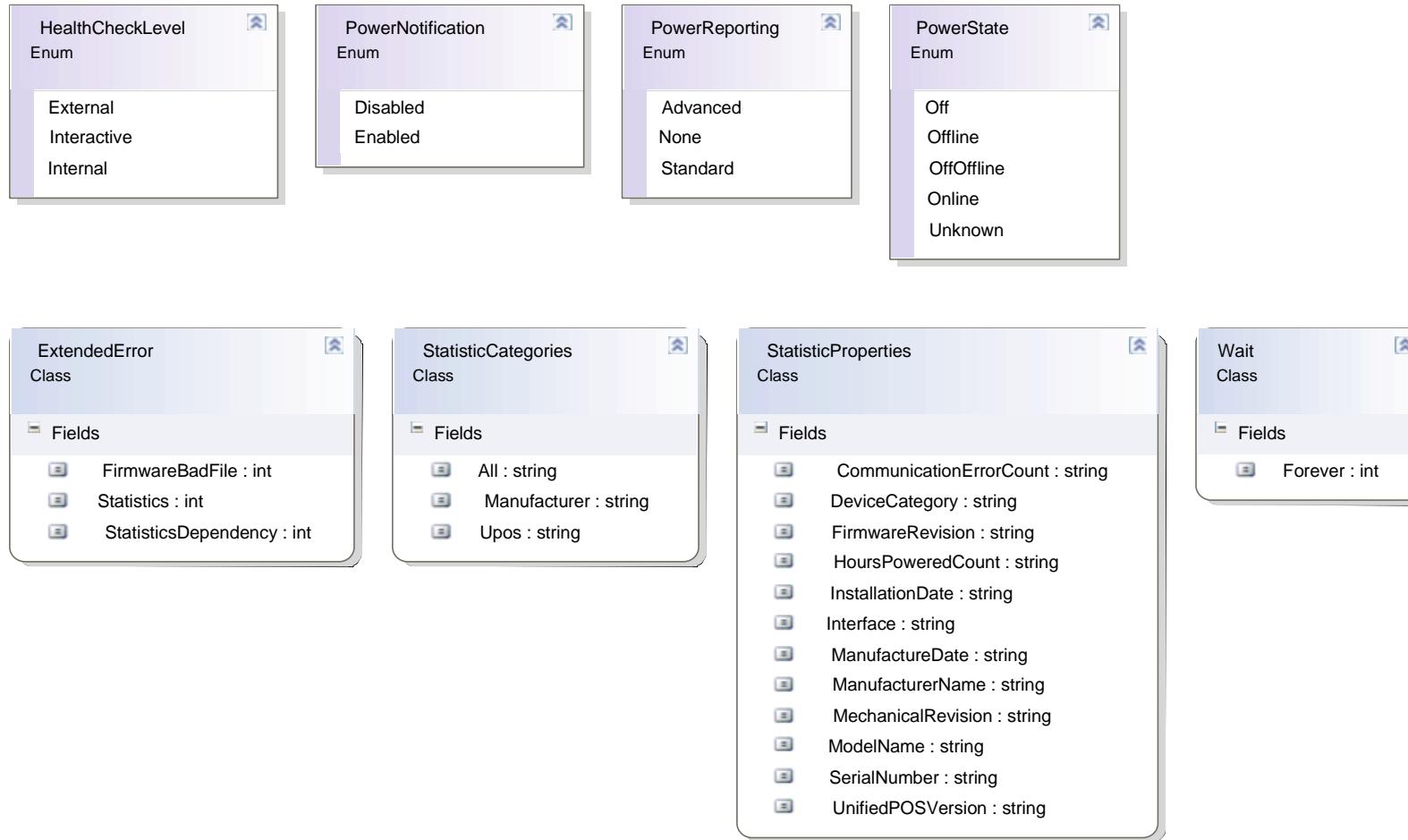
WS-POS 1.1 Technical Specification

Coin Dispenser

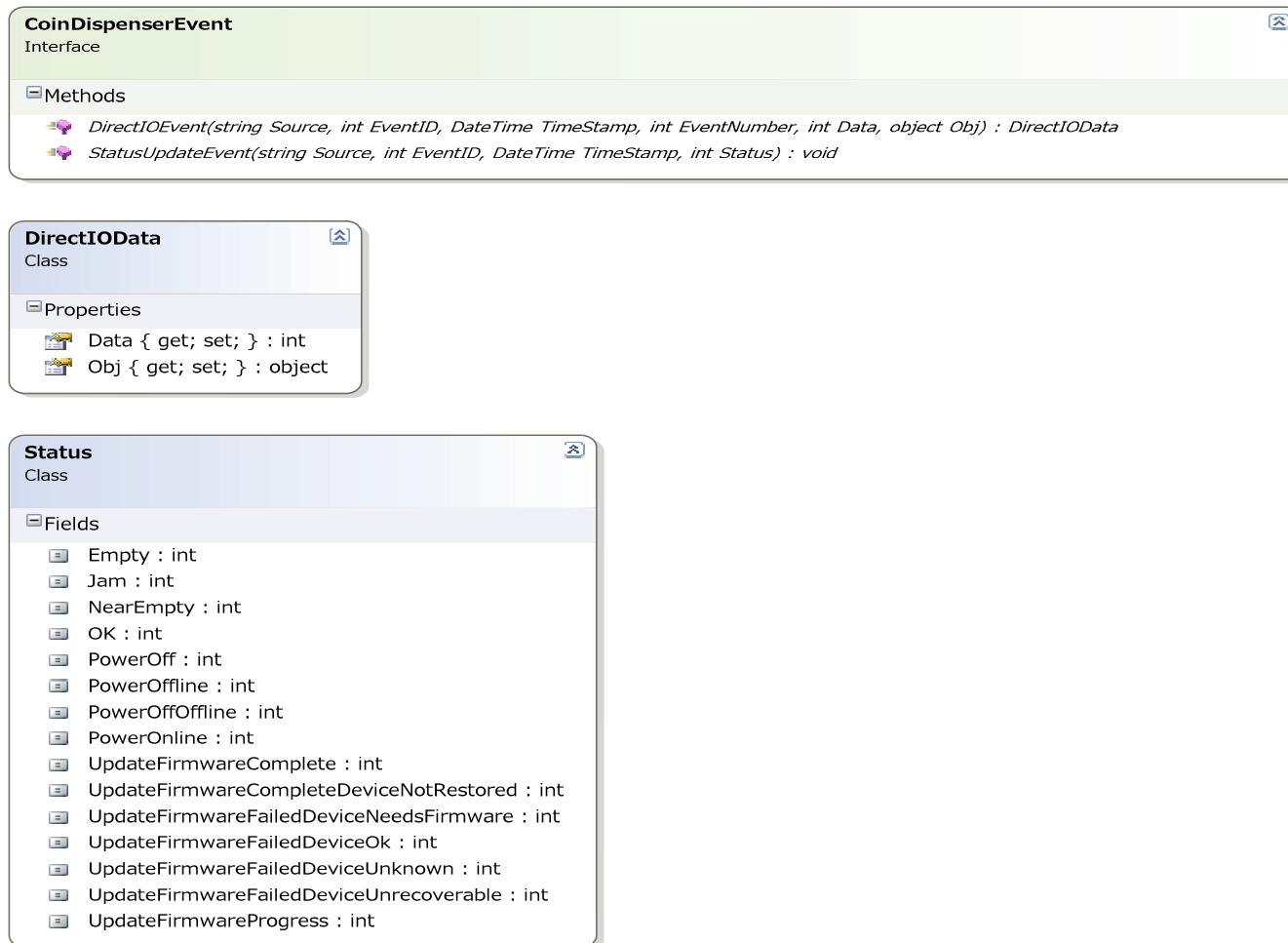


WS-POS 1.1 Technical Specification

Coin Dispenser



Coin Dispenser Events



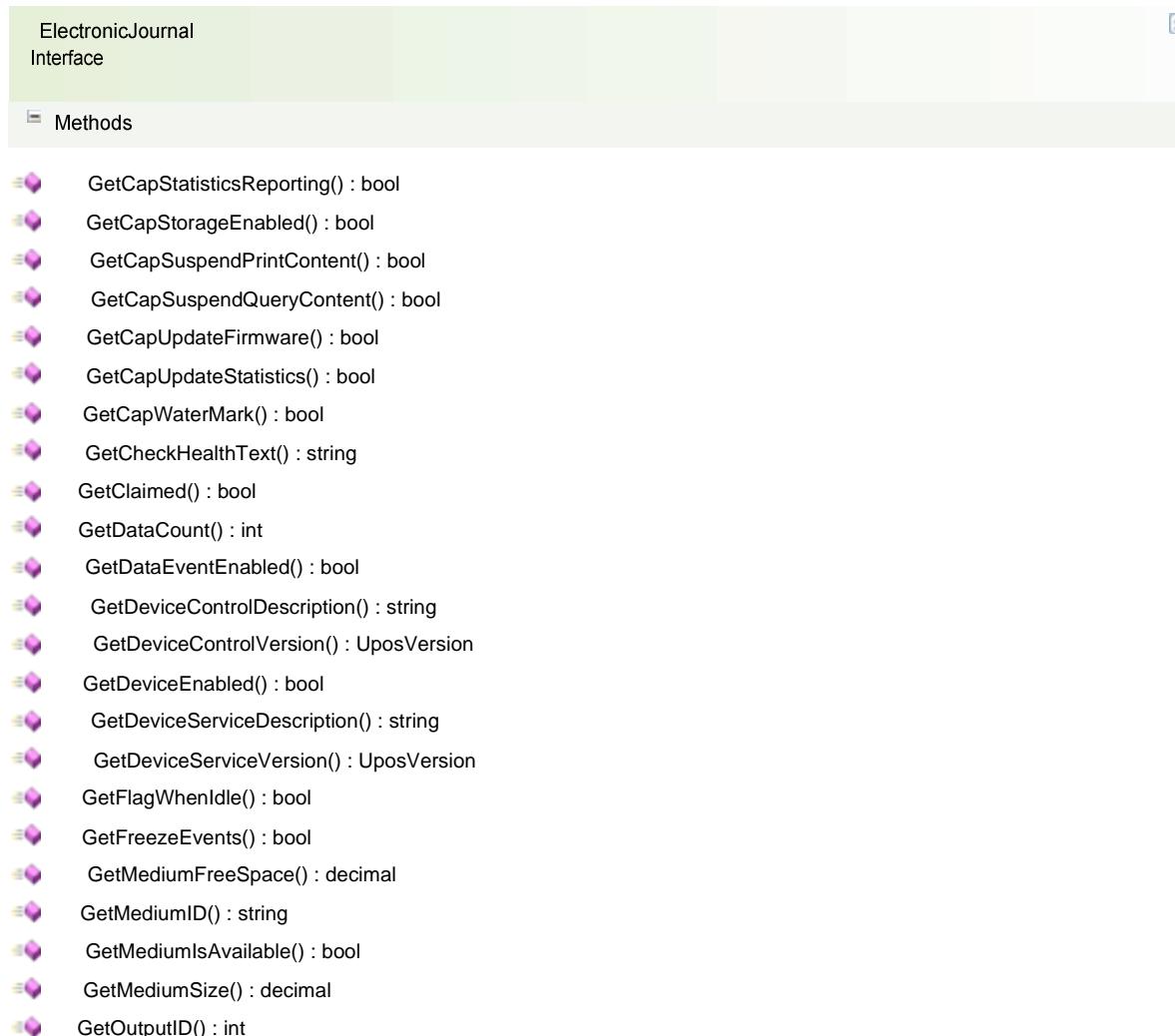
Electronic Journal

ElectronicJournal
Interface

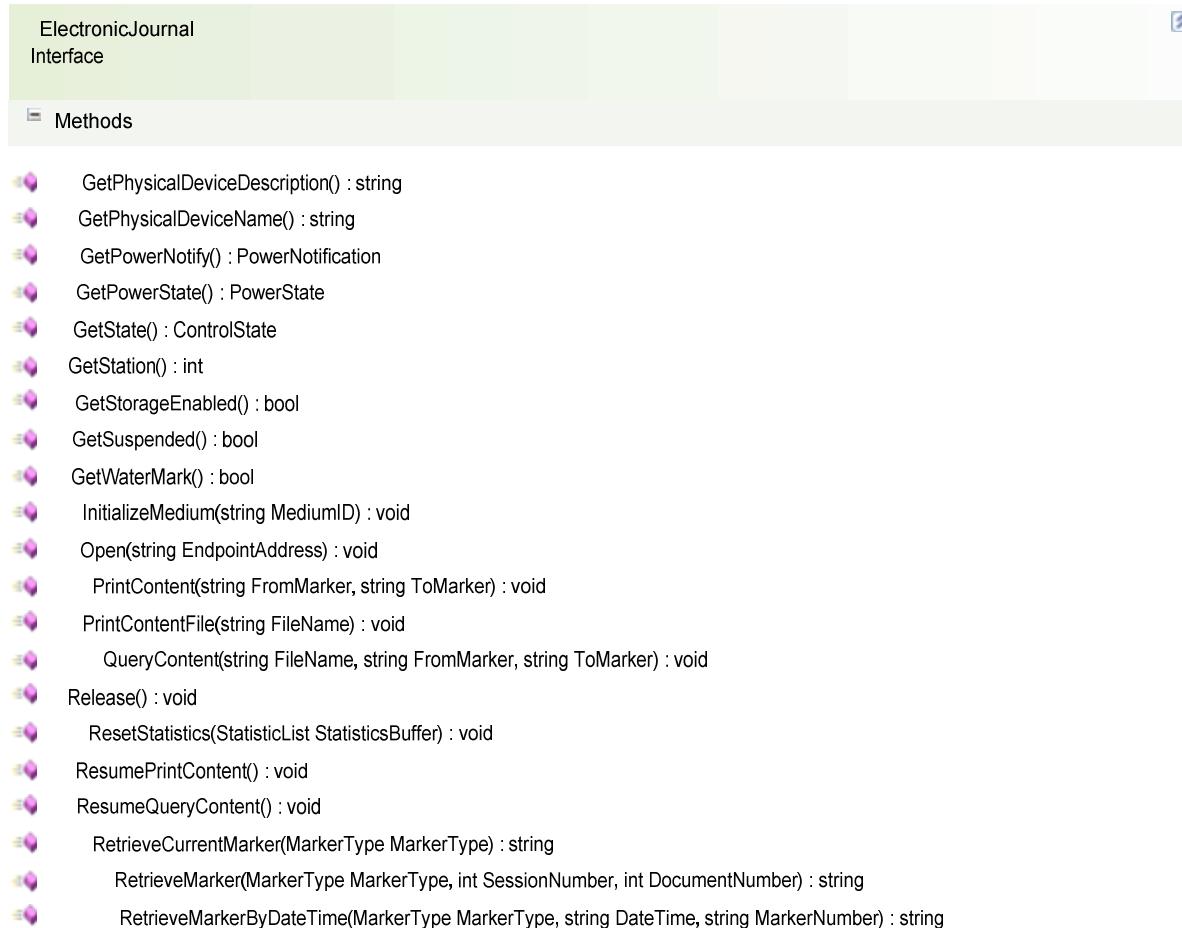
The diagram shows a UML class named "ElectronicJournal" with a green header bar. Below the header, a section titled "Methods" is expanded, revealing a list of 34 methods. Each method is preceded by a small icon consisting of a blue square with a white circle and a purple square with a white circle.

- >AddMarker(string Marker) : void
- CancelPrintContent() : void
- CancelQueryContent() : void
- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- ClearInput() : void
- ClearOutput() : void
- Close(string EndpointAddress) : void
 - CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
- DirectIO(int Command, int Data, object Obj) : DirectIOData
- EraseMedium() : void
- GetAsyncMode() : bool
- GetAutoDisable() : bool
- GetCapAddMarker() : bool
- GetCapCompareFirmwareVersion() : bool
- GetCapErasableMedium() : bool
- GetCapInitializeMedium() : bool
- GetCapMediumIsAvailable() : bool
- GetCapPowerReporting() : PowerReporting
- GetCapPrintContent() : bool
- GetCapPrintContentFile() : bool
- GetCapRetrieveCurrentMarker() : bool
- GetCapRetrieveMarker() : bool
- GetCapRetrieveMarkerByDateTime() : bool
- GetCapRetrieveMarkersDateTime() : bool
- GetCapStation() : int

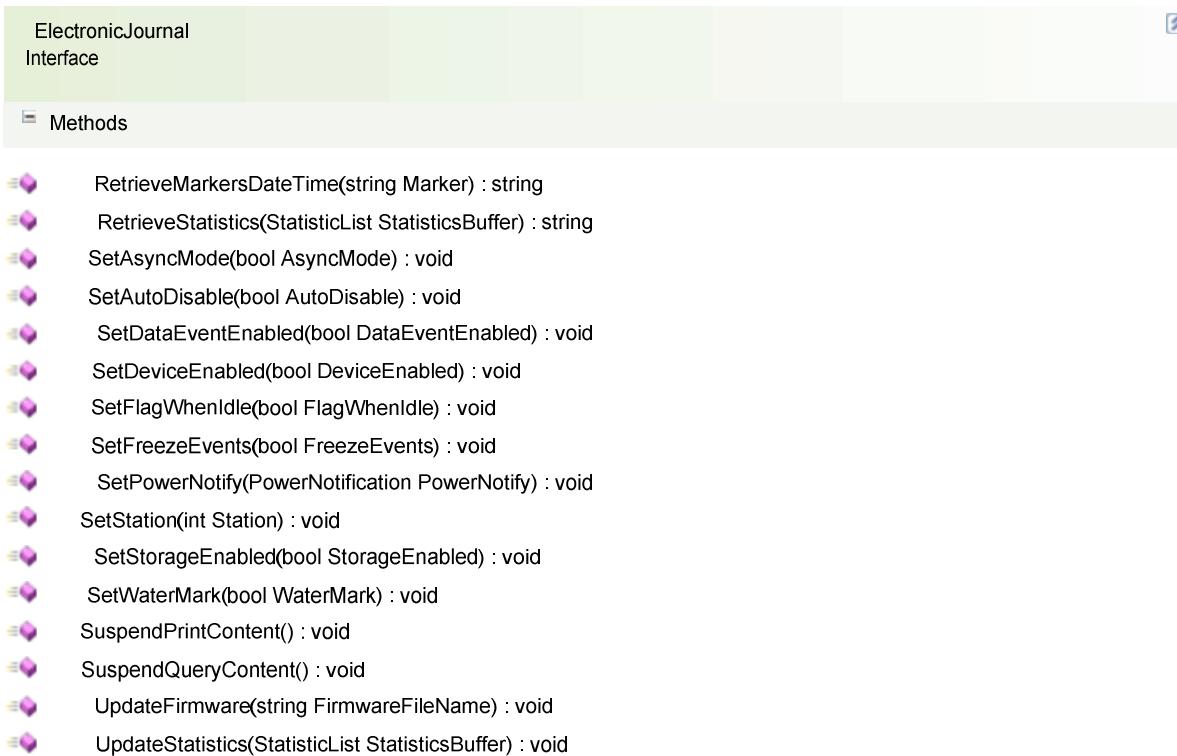
Electronic Journal



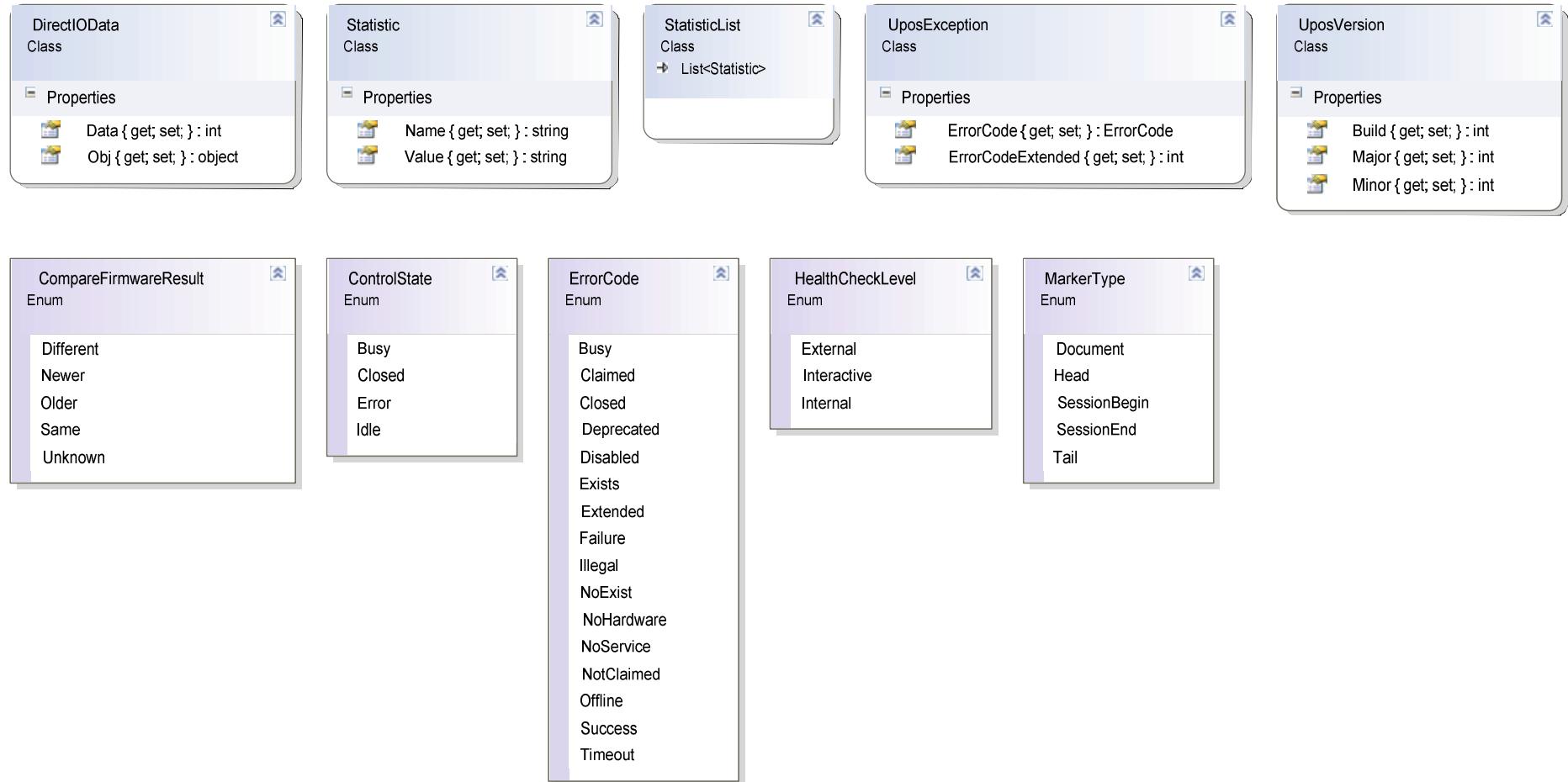
Electronic Journal



Electronic Journal

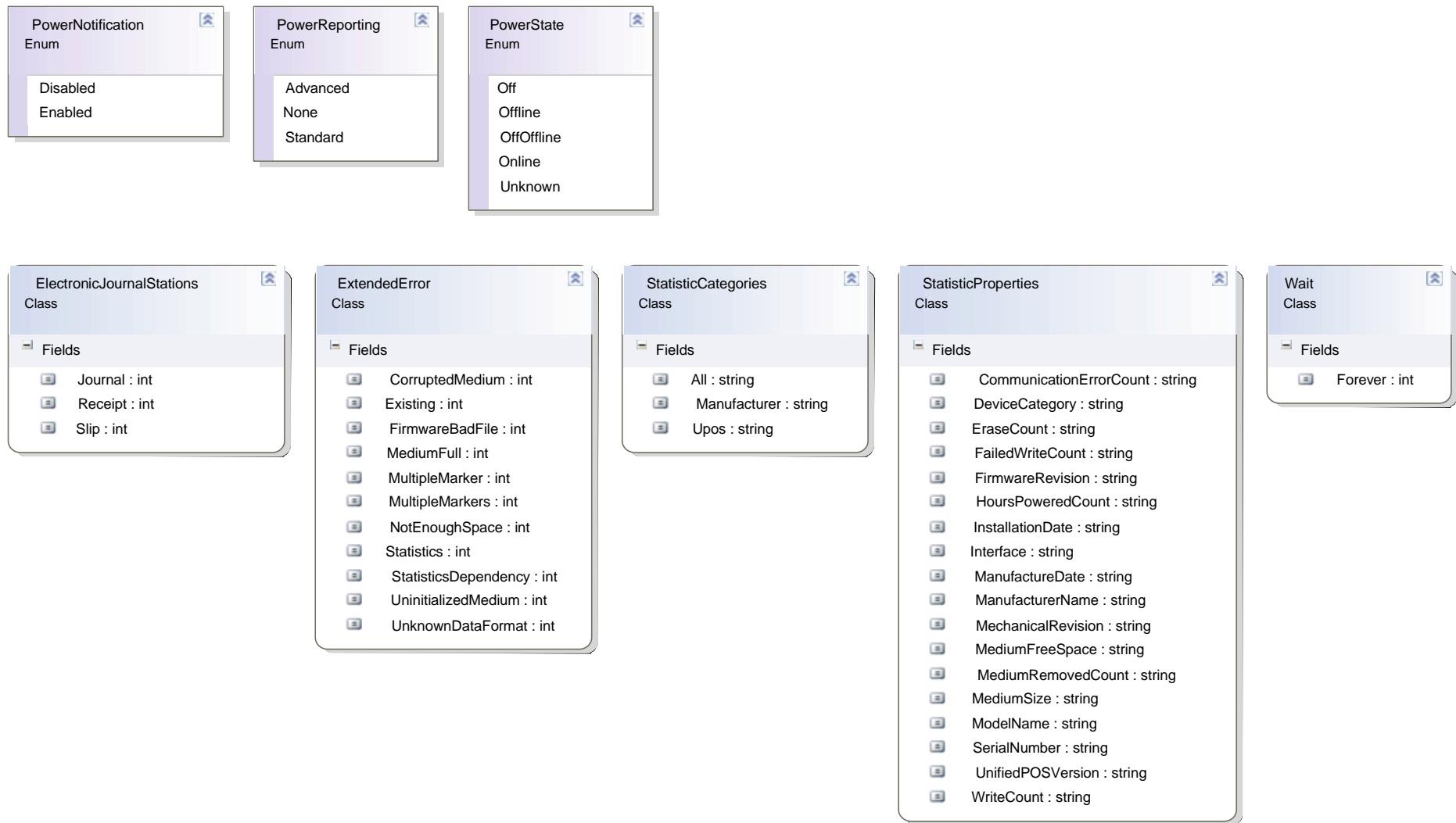


Electronic Journal

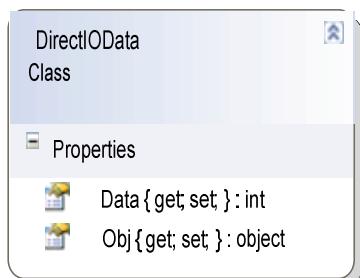
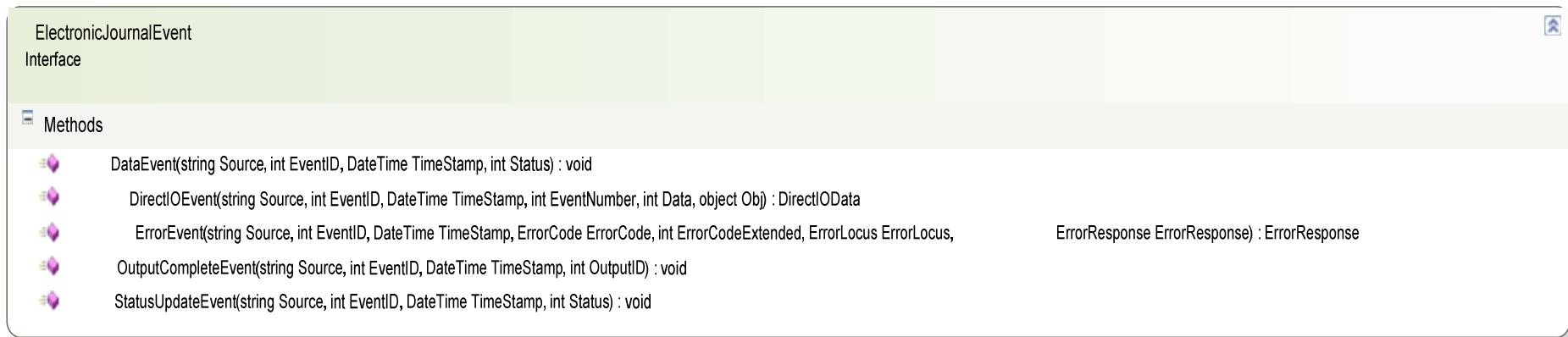


WS-POS 1.1 Technical Specification

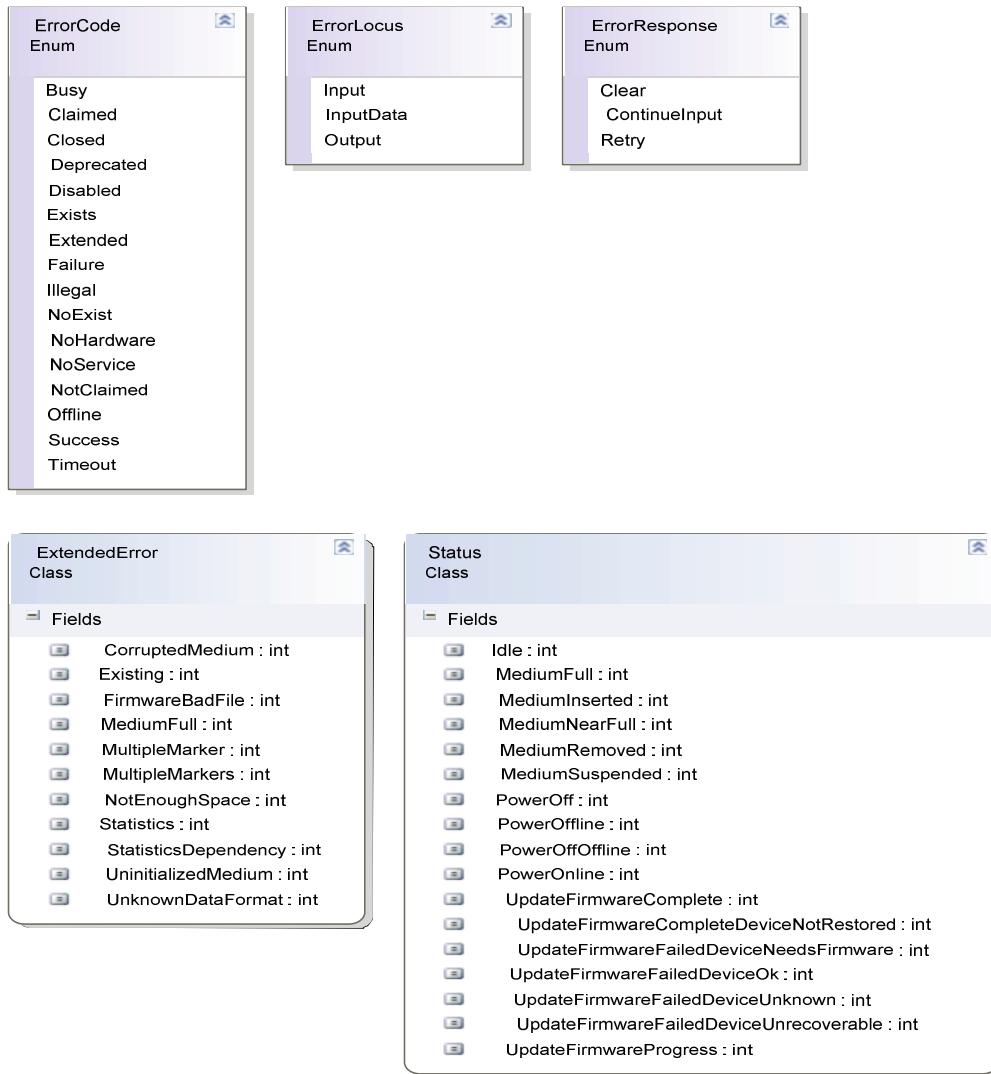
Electronic Journal



Electronic Journal Events



Electronic Journal Events



Electronic Value R/W (Electronic Value Reader and Writer)

ElectronicValueRW
Interface

Methods

- AccessLog(int SequenceNumber, TransactionLogType Type, int Timeout) : void
- ActivateService(int Data, object Obj) : EVRWResult
- AddValue(int SequenceNumber, int Timeout) : void
- BeginDetection(BeginDetectionType Type, int Timeout) : void
- BeginRemoval(int Timeout) : void
- CancelValue(int SequenceNumber, int Timeout) : void
- CaptureCard() : void
- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- ClearInput() : void
- ClearInputProperties() : void
- ClearOutput() : void
- Close(string EndpointAddress) : void
 - CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
- DirectIO(int Command, int Data, object Obj) : DirectIOData
- EndDetection() : void
- EndRemoval() : void
- EnumerateCardServices() : void
- GetAccountNumber() : string
- GetAdditionalSecurityInformation() : string
- GetAmount() : decimal
- GetApprovalCode() : string
- GetAsyncMode() : bool
- GetAutoDisable() : bool
- GetBalance() : decimal
- GetBalanceOfPoint() : decimal
- GetCapActivateService() : bool
- GetCapAddValue() : bool

Electronic Value R/W (Electronic Value Reader and Writer)



Methods

- GetCapCancelValue() : bool
- GetCapCardSensor() : int
- GetCapCompareFirmwareVersion() : bool
- GetCapDetectionControl() : int
- GetCapElectronicMoney() : bool
- GetCapEnumerateCardServices() : bool
- GetCapIndirectTransactionLog() : bool
- GetCapLockTerminal() : bool
- GetCapLogStatus() : bool
- GetCapMediumID() : bool
- GetCapPoint() : bool
 - GetCapPowerReporting() : PowerReporting
 - GetCapStatisticsReporting() : bool
 - GetCapSubtractValue() : bool
 - GetCapTransaction() : bool
 - GetCapTransactionLog() : bool
 - GetCapUnlockTerminal() : bool
 - GetCapUpdateFirmware() : bool
 - GetCapUpdateKey() : bool
 - GetCapUpdateStatistics() : bool
 - GetCapVoucher() : bool
 - GetCapWriteValue() : bool
 - GetCardServiceList() : CardServiceList
 - GetCheckHealthText() : string
- GetClaimed() : bool
- GetCurrentService() : string
- GetDataCount() : int
- GetDataEventEnabled() : bool
- GetDetectionControl() : bool

Electronic Value R/W (Electronic Value Reader and Writer)



Methods

- » GetDetectionStatus() : DetectionState
- » GetDeviceControlDescription() : string
- » GetDeviceControlVersion() : UpoSVersion
- » GetDeviceEnabled() : bool
- » GetDeviceServiceDescription() : string
- » GetDeviceServiceVersion() : UpoSVersion
- » GetExpirationDate() : DateTime
- » GetFreezeEvents() : bool
- » GetLastUsedDate() : DateTime
- » GetLogStatus() : LogState
- » GetMediumID() : string
- » GetOutputID() : int
- » GetPhysicalDeviceDescription() : string
- » GetPhysicalDeviceName() : string
- » GetPoint() : decimal
- » GetPowerNotify() : PowerNotification
- » GetPowerState() : PowerState
- » GetReaderWriterServiceList() : ReaderWriterServiceList
- » GetSequenceNumber() : int
- » GetSettledAmount() : decimal
- » GetSettledPoint() : decimal
- » GetState() : ControlState
- » GetTransactionLog() : string
- » GetVoucherID() : VoucherID
- » GetVoucherIDList() : VoucherIDList
- » LockTerminal() : void
- » Open(string EndpointAddress) : void
- » ReadValue(int SequenceNumber, int Timeout) : void
- » Release() : void
- » ResetStatistics(StatisticList StatisticsBuffer) : void

Electronic Value R/W (Electronic Value Reader and Writer)

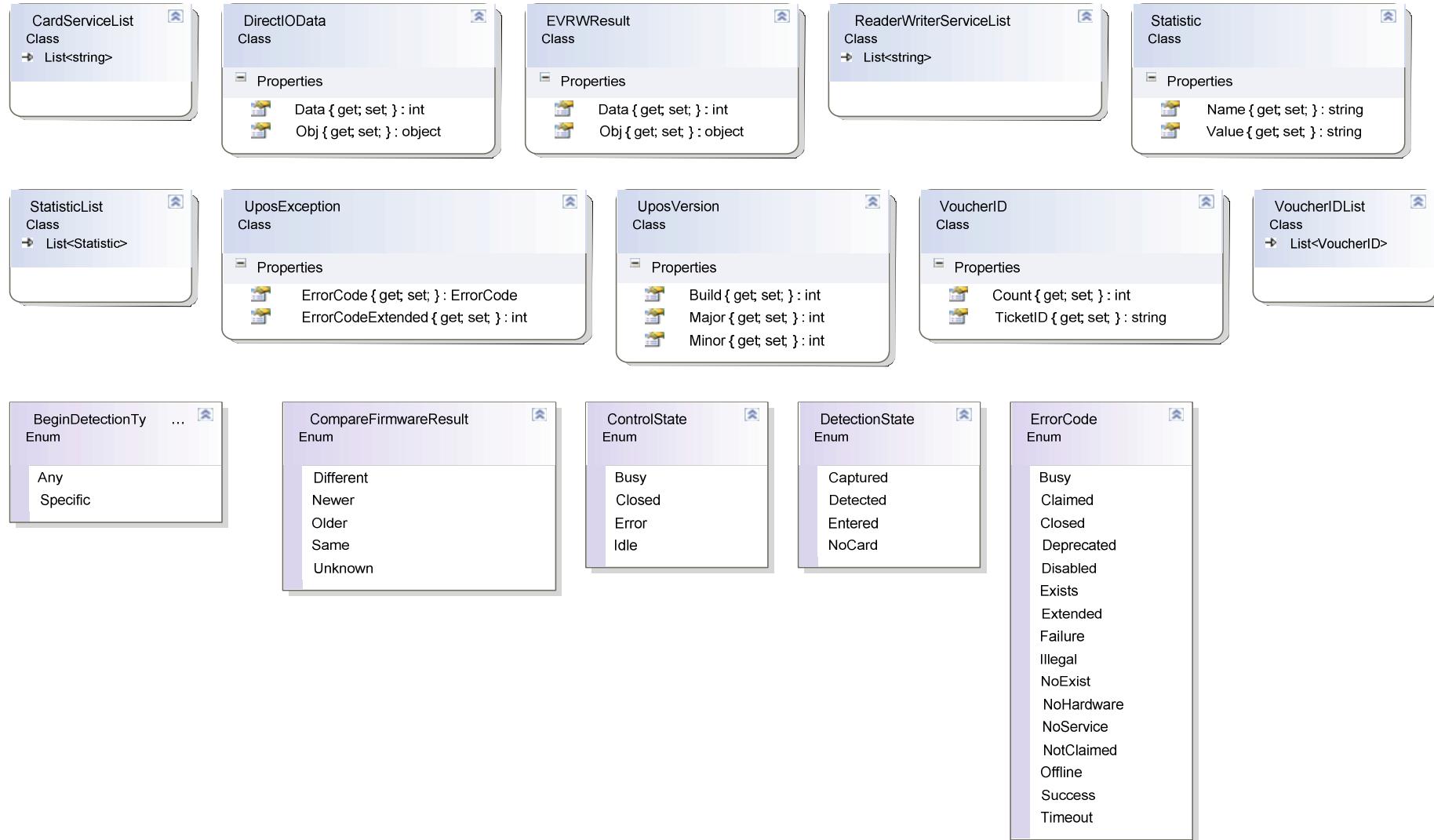


Methods

- RetrieveStatistics(StatisticList StatisticsBuffer) : string
- SetAdditionalSecurityInformation(string AdditionalSecurityInformation) : void
- SetAmount(decimal Amount) : void
- SetApprovalCode(string ApprovalCode) : void
- SetAsyncMode(bool AsyncMode) : void
- SetAutoDisable(bool AutoDisable) : void
- SetCurrentService(string CurrentService) : void
- SetDataEventEnabled(bool DataEventEnabled) : void
- SetDetectionControl(bool DetectionControl) : void
- SetDeviceEnabled(bool DeviceEnabled) : void
- SetFreezeEvents(bool FreezeEvents) : void
- SetMediumID(string MediumID) : void
- SetPoint(decimal Point) : void
- SetPowerNotify(PowerNotification PowerNotify) : void
- SetVoucherID(VoucherID VoucherID) : void
- SetVoucherIDList(VoucherIDList VoucherIDList) : void
- SubtractValue(int SequenceNumber, int Timeout) : void
- TransactionAccess(TransactionControl Control) : void
- UnlockTerminal() : void
- UpdateFirmware(string FirmwareFileName) : void
- UpdateKey(int Data, object Obj) : EVRWRResult
- UpdateStatistics(StatisticList StatisticsBuffer) : void
- WriteValue(int SequenceNumber, int Timeout) : void

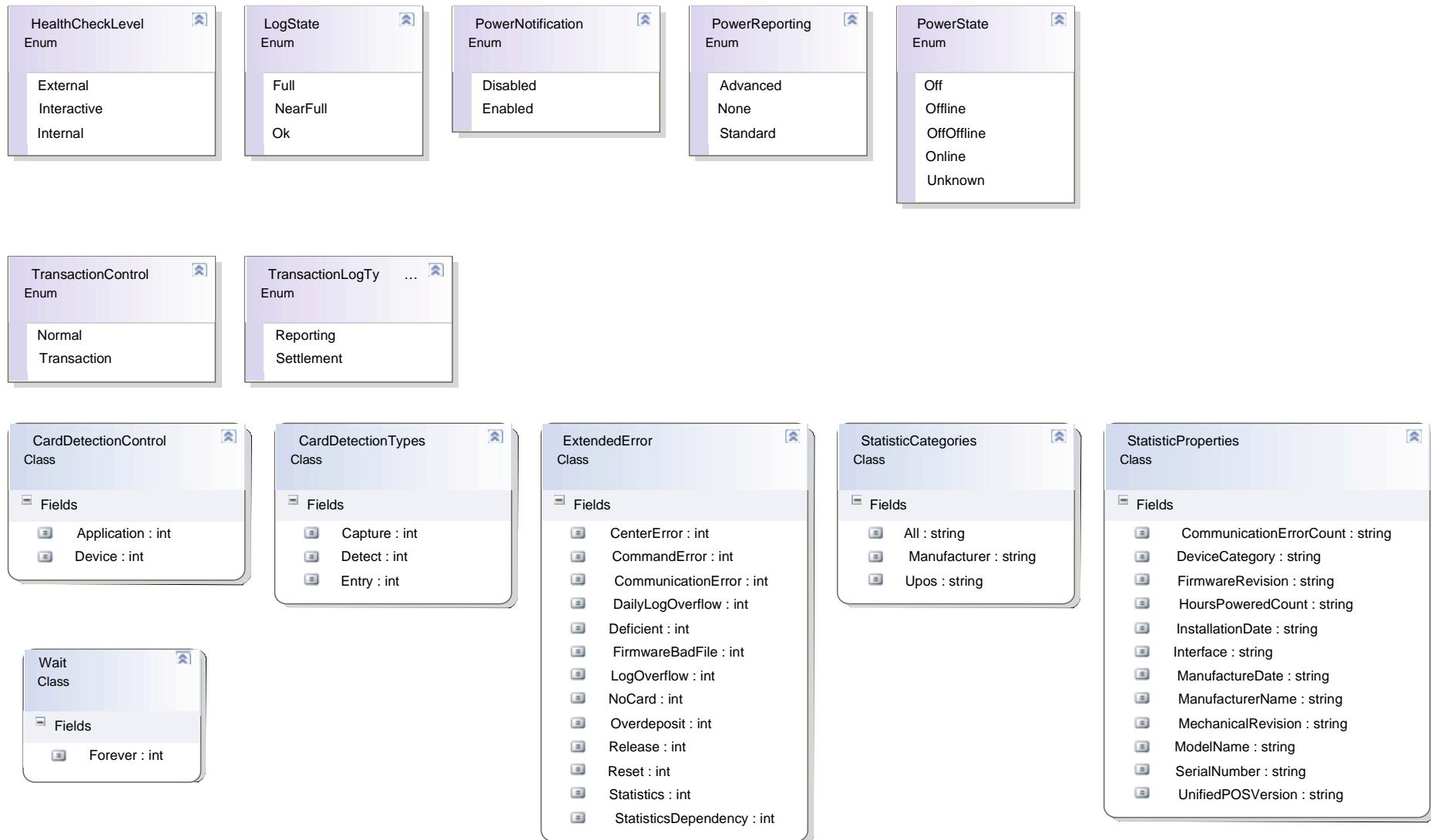
WS-POS 1.1 Technical Specification

Electronic Value R/W (Electronic Value Reader and Writer)

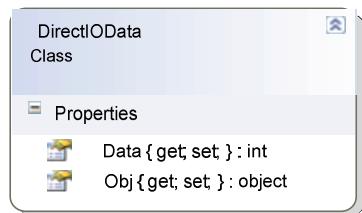
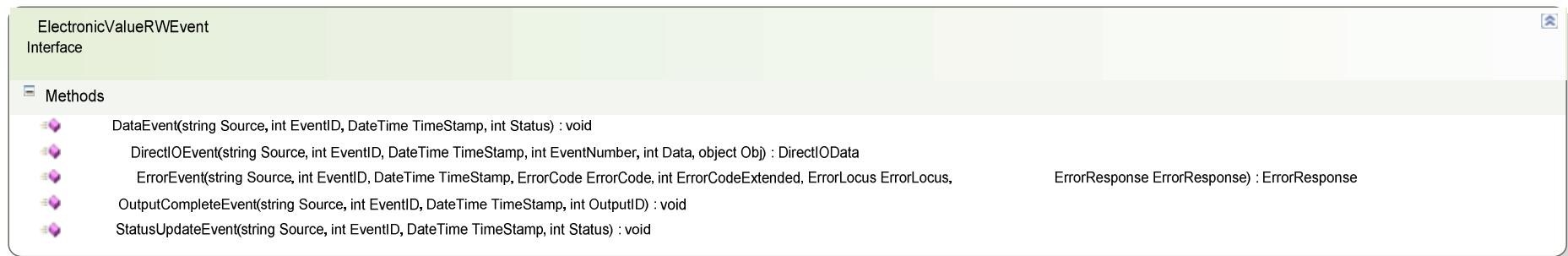


WS-POS 1.1 Technical Specification

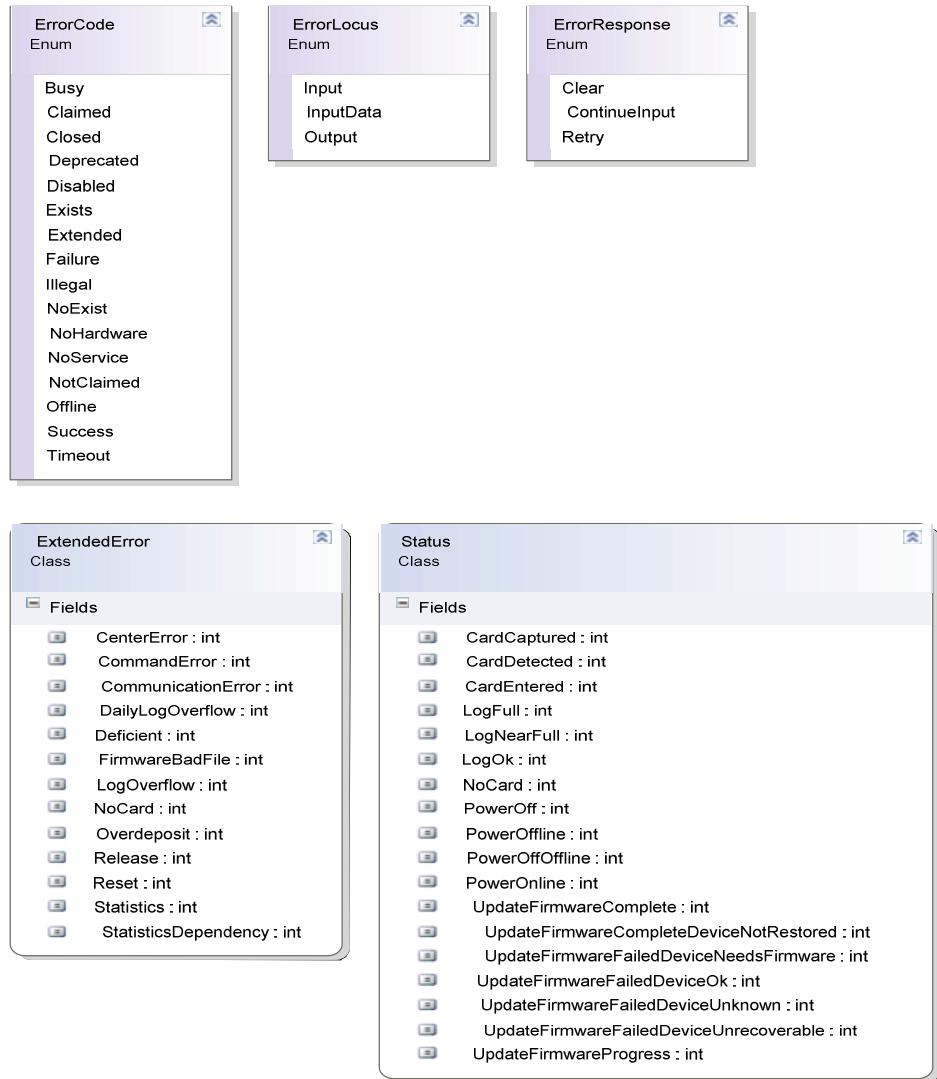
Electronic Value R/W (Electronic Value Reader and Writer)



Electronic Value R/W (Electronic Value Reader and Writer) Events



Electronic Value R/W (Electronic Value Reader and Writer) Events



Fiscal Printer

The screenshot shows a software interface for the 'FiscalPrinter Interface'. The interface has a light green header bar with the title. Below it is a large white area containing a list of methods. A sidebar on the left lists categories like 'Methods' and 'Properties'. The 'Methods' category is expanded, showing 30 methods, each preceded by a small purple icon.

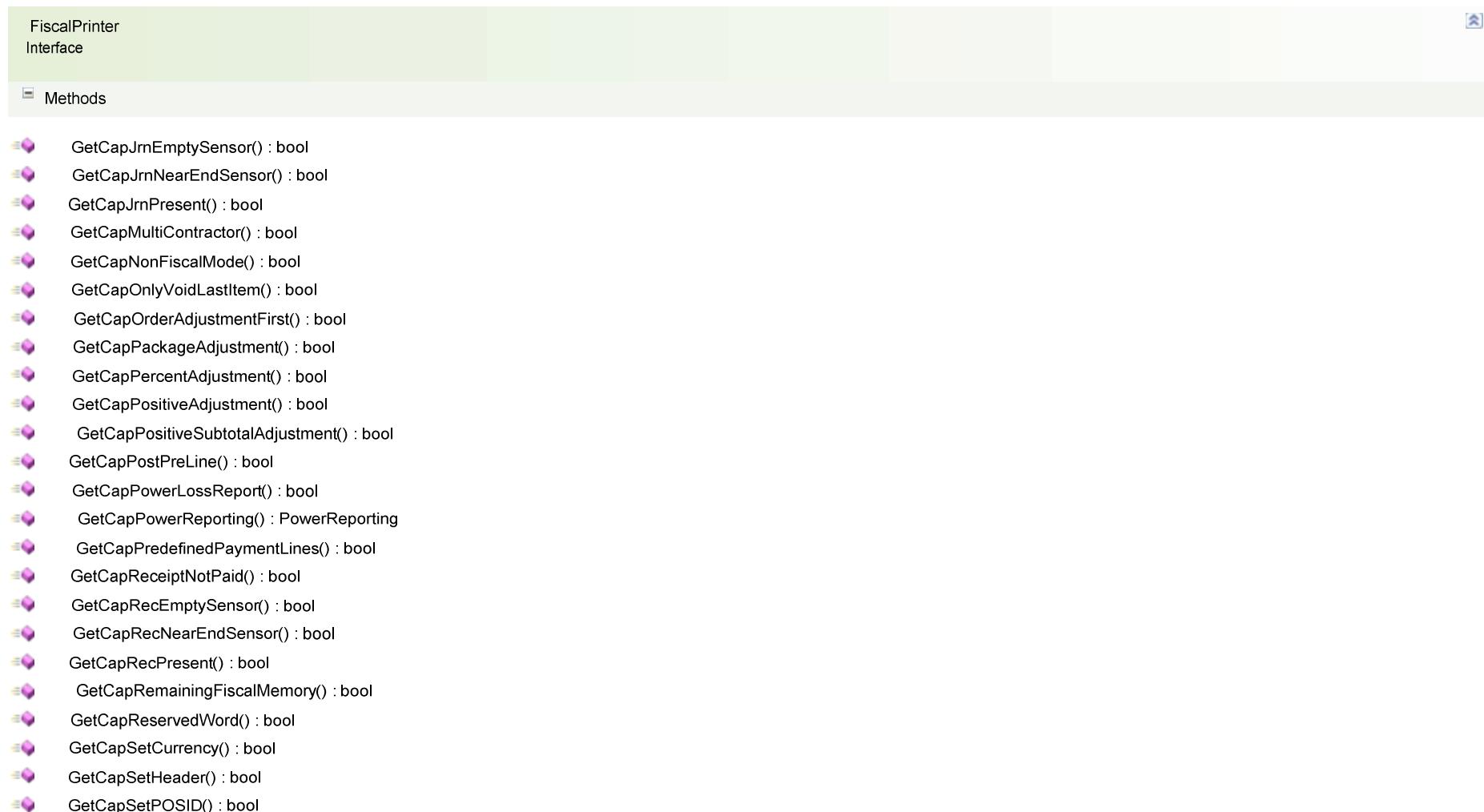
- BeginInitDocument(int DocumentAmount) : void
- BeginInitReceipt(bool PrintHeader) : void
- BeginInitFixedOutput(FiscalReceiptStation Station, int DocumentType) : void
- BeginInitInsertion(int Timeout) : void
- BeginInitItemList(int VatID) : void
- BeginInitNonFiscal() : void
- BeginInitRemoval(int Timeout) : void
- BeginInitTraining() : void
- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- ClearError() : void
- ClearOutput() : void
- Close(string EndpointAddress) : void
 - CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
- DirectIO(int Command, int Data, object Obj) : DirectIOData
- EndBeginInitDocument() : void
- EndBeginInitReceipt(bool PrintHeader) : void
- EndBeginInitFixedOutput() : void
- EndBeginInitInsertion() : void
- EndBeginInitItemList() : void
- EndBeginInitNonFiscal() : void
- EndBeginInitRemoval() : void
- EndBeginInitTraining() : void
- GetActualCurrency() : FiscalCurrency

Fiscal Printer

The screenshot shows a software interface for the 'FiscalPrinter Interface'. The 'Methods' section is expanded, displaying a list of methods:

- GetAdditionalHeader() : string
- GetAdditionalTrailer() : string
- GetAmountDecimalPlaces() : int
- GetAsyncMode() : bool
- GetCapAdditionalHeader() : bool
- GetCapAdditionalLines() : bool
- GetCapAdditionalTrailer() : bool
- GetCapAmountAdjustment() : bool
- GetCapChangeDue() : bool
- GetCapCheckTotal() : bool
- GetCapCompareFirmwareVersion() : bool
- GetCapCoverSensor() : bool
- GetCapDoubleWidth() : bool
- GetCapDuplicateReceipt() : bool
- GetCapEmptyReceiptIsVoidable() : bool
- GetCapFiscalReceiptStation() : bool
- GetCapFiscalReceiptType() : bool
- GetCapFixedOutput() : bool
- GetCapHasVatTable() : bool
- GetCapIndependentHeader() : bool
- GetCapItemList() : bool

Fiscal Printer



Fiscal Printer

FiscalPrinter
Interface

Methods

- ≡ GetCapSetStoreFiscalID() : bool
- ≡ GetCapSetTrailer() : bool
- ≡ GetCapSetVatTable() : bool
- ≡ GetCapSlpEmptySensor() : bool
- ≡ GetCapSlpFiscalDocument() : bool
- ≡ GetCapSlpFullSlip() : bool
- ≡ GetCapSlpNearEndSensor() : bool
- ≡ GetCapSlpPresent() : bool
- ≡ GetCapSlpValidation() : bool
- ≡ GetCapStatisticsReporting() : bool
- ≡ GetCapSubAmountAdjustment() : bool
- ≡ GetCapSubPercentAdjustment() : bool
- ≡ GetCapSubtotal() : bool
- ≡ GetCapTotalizerType() : bool
- ≡ GetCapTrainingMode() : bool
- ≡ GetCapUpdateFirmware() : bool
- ≡ GetCapUpdateStatistics() : bool
- ≡ GetCapValidateJournal() : bool
- ≡ GetCapXReport() : bool
- ≡ GetChangeDue() : string
- ≡ GetCheckHealthText() : string
- ≡ GetCheckTotal() : bool
- ≡ GetClaimed() : bool
- ≡ GetContractorID() : FiscalContractorID

Fiscal Printer

FiscalPrinter
Interface

Methods

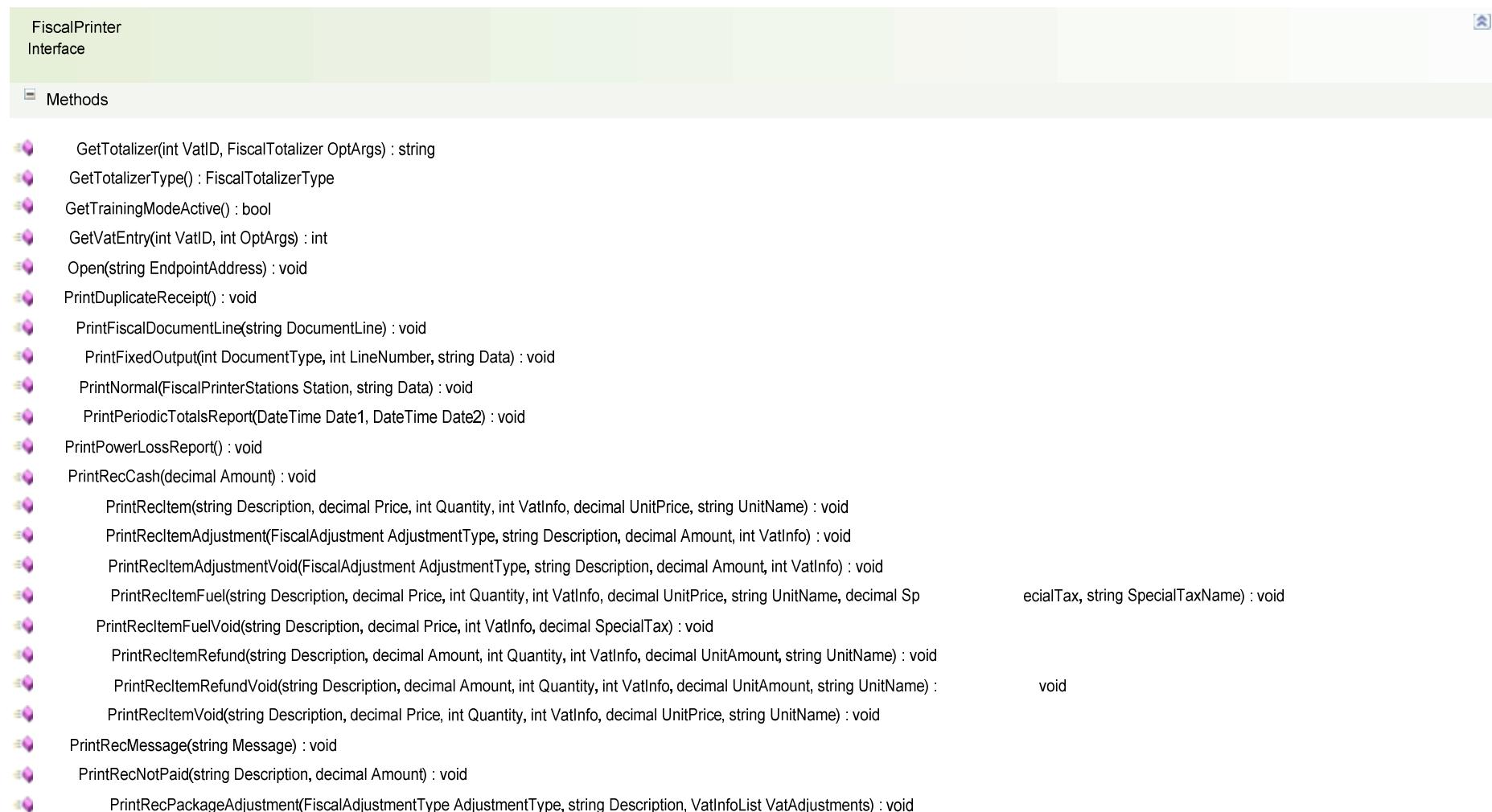
- GetCountryCode() : int
- GetCoverOpen() : bool
- GetData(FiscalData DataItem, int OptArgs) : FiscalDataItem
- GetDate() : DateTime
- GetDateType() : FiscalDateType
- GetDayOpened() : bool
- GetDescriptionLength() : int
- GetDeviceControlDescription() : string
- GetDeviceControlVersion() : UposVersion
- GetDeviceEnabled() : bool
- GetDeviceServiceDescription() : string
- GetDeviceServiceVersion() : UposVersion
- GetDuplicateReceipt() : bool
- GetErrorLevel() : FiscalErrorLevel
- GetErrorOutID() : int
- GetErrorState() : FiscalPrinterState
- GetErrorStation() : FiscalPrinterStations
- GetErrorString() : string
- GetFiscalReceiptStation() : FiscalReceiptStation
- GetFiscalReceiptType() : FiscalReceiptType
- GetFlagWhenIdle() : bool
- GetFreezeEvents() : bool

Fiscal Printer

The diagram shows a UML class named 'FiscalPrinter' with the 'Interface' keyword above it. A single method, 'Methods', is listed under the interface. The 'Methods' section contains 30 methods, each preceded by a small icon.

- ≡ GetJrnEmpty() : bool
- ≡ GetJrnNearEnd() : bool
- ≡ GetMessageLength() : int
- ≡ GetMessageType() : FiscalMessageType
- ≡ GetNumHeaderLines() : int
- ≡ GetNumTrailerLines() : int
- ≡ GetNumVatRates() : int
- ≡ GetOutputID() : int
- ≡ GetPhysicalDeviceDescription() : string
- ≡ GetPhysicalDeviceName() : string
- ≡ GetPostLine() : string
- ≡ GetPowerNotify() : PowerNotification
- ≡ GetPowerState() : PowerState
- ≡ GetPredefinedPaymentLines() : PaymentLinesList
- ≡ GetPreLine() : string
- ≡ GetPrinterState() : FiscalPrinterState
- ≡ GetQuantityDecimalPlaces() : int
- ≡ GetQuantityLength() : int
- ≡ GetRecEmpty() : bool
- ≡ GetRecNearEnd() : bool
- ≡ GetRemainingFiscalMemory() : int
- ≡ GetReservedWord() : string
- ≡ GetSlipSelection() : FiscalSlipSelection
- ≡ GetSlpEmpty() : bool
- ≡ GetSlpNearEnd() : bool
- ≡ GetState() : ControlState

Fiscal Printer



FiscalPrinter
Interface

Methods

- ⊕ GetTotalizer(int VatID, FiscalTotalizer OptArgs) : string
- ⊕ GetTotalizerType() : FiscalTotalizerType
- ⊕ GetTrainingModeActive() : bool
- ⊕ GetVatEntry(int VatID, int OptArgs) : int
- ⊕ Open(string EndpointAddress) : void
- ⊕ PrintDuplicateReceipt() : void
- ⊕ PrintFiscalDocumentLine(string DocumentLine) : void
- ⊕ PrintFixedOutput(int DocumentType, int LineNumber, string Data) : void
- ⊕ PrintNormal(FiscalPrinterStations Station, string Data) : void
- ⊕ PrintPeriodicTotalsReport(DateTime Date1, DateTime Date2) : void
- ⊕ PrintPowerLossReport() : void
- ⊕ PrintRecCash(decimal Amount) : void
 - ⊕ PrintRecItem(string Description, decimal Price, int Quantity, int VatInfo, decimal UnitPrice, string UnitName) : void
 - ⊕ PrintRecItemAdjustment(FiscalAdjustment AdjustmentType, string Description, decimal Amount, int VatInfo) : void
 - ⊕ PrintRecItemAdjustmentVoid(FiscalAdjustment AdjustmentType, string Description, decimal Amount, int VatInfo) : void
 - ⊕ PrintRecItemFuel(string Description, decimal Price, int Quantity, int VatInfo, decimal UnitPrice, string UnitName, decimal SpecialTax, string SpecialTaxName) : void
 - ⊕ PrintRecItemFuelVoid(string Description, decimal Price, int VatInfo, decimal SpecialTax) : void
 - ⊕ PrintRecItemRefund(string Description, decimal Amount, int Quantity, int VatInfo, decimal UnitAmount, string UnitName) : void
 - ⊕ PrintRecItemRefundVoid(string Description, decimal Amount, int Quantity, int VatInfo, decimal UnitAmount, string UnitName) : void
 - ⊕ PrintRecItemVoid(string Description, decimal Price, int Quantity, int VatInfo, decimal UnitPrice, string UnitName) : void
- ⊕ PrintRecMessage(string Message) : void
- ⊕ PrintRecNotPaid(string Description, decimal Amount) : void
- ⊕ PrintRecPackageAdjustment(FiscalAdjustmentType AdjustmentType, string Description, VatInfoList VatAdjustments) : void

Fiscal Printer

FiscalPrinter
Interface

Methods

- PrintRecPackageAdjustVoid(FiscalAdjustmentType AdjustmentType, VatInfoList VatAdjustments) : void
- PrintRecRefund(string Description, decimal Amount, int VatInfo) : void
- PrintRecRefundVoid(string Description, decimal Amount, int VatInfo) : void
- PrintRecSubtotal(decimal Amount) : void
- PrintRecSubtotalAdjustment(FiscalAdjustment AdjustmentType, string Description, decimal Amount) : void
- PrintRecSubtotalAdjustVoid(FiscalAdjustment AdjustmentType, decimal Amount) : void
- PrintRecTaxID(string TaxID) : void
- PrintRecTotal(decimal Total, decimal Payment, string Description) : void
- PrintRecVoid(string Description) : void
- PrintReport(ReportType ReportType, string StartNum, string EndNum) : void
- PrintXReport() : void
- PrintZReport() : void
- Release() : void
- ResetPrinter() : void
- ResetStatistics(StatisticList StatisticsBuffer) : void
- RetrieveStatistics(StatisticList StatisticsBuffer) : string
- SetAdditionalHeader(string AdditionalHeader) : void
- SetAdditionalTrailer(string AdditionalTrailer) : void
- SetAsyncMode(bool AsyncMode) : void
- SetChangeDue(string ChangeDue) : void
- SetCheckTotal(bool CheckTotal) : void
- SetContractorID(FiscalContractorID ContractorID) : void
- SetCurrency(FiscalCurrency NewCurrency) : void
- SetDate(DateTime Date) : void
- SetDateType(FiscalDateType DateType) : void
- SetDeviceEnabled(bool DeviceEnabled) : void
- SetDuplicateReceipt(bool DuplicateReceipt) : void

Fiscal Printer

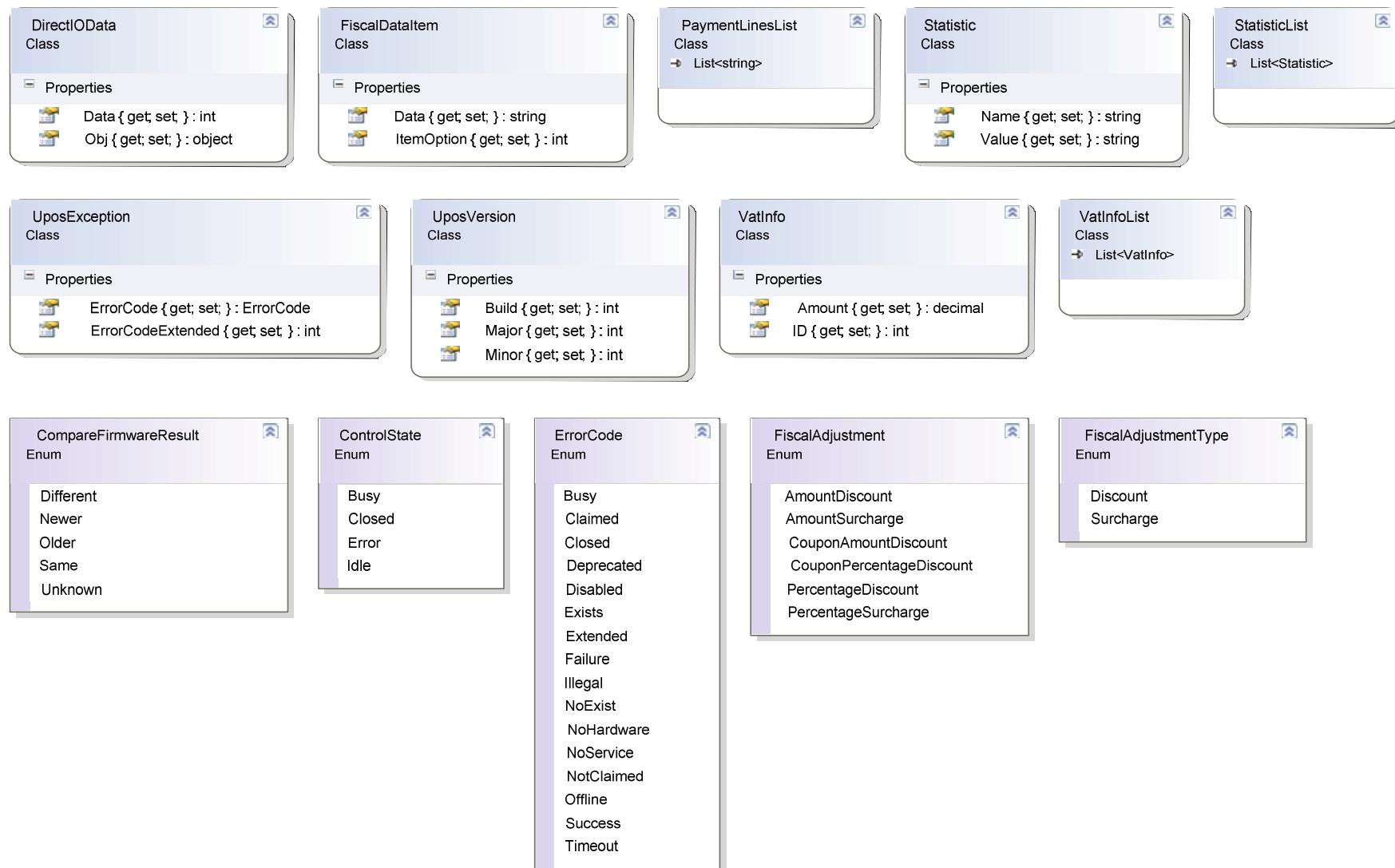
FiscalPrinter
Interface

Methods

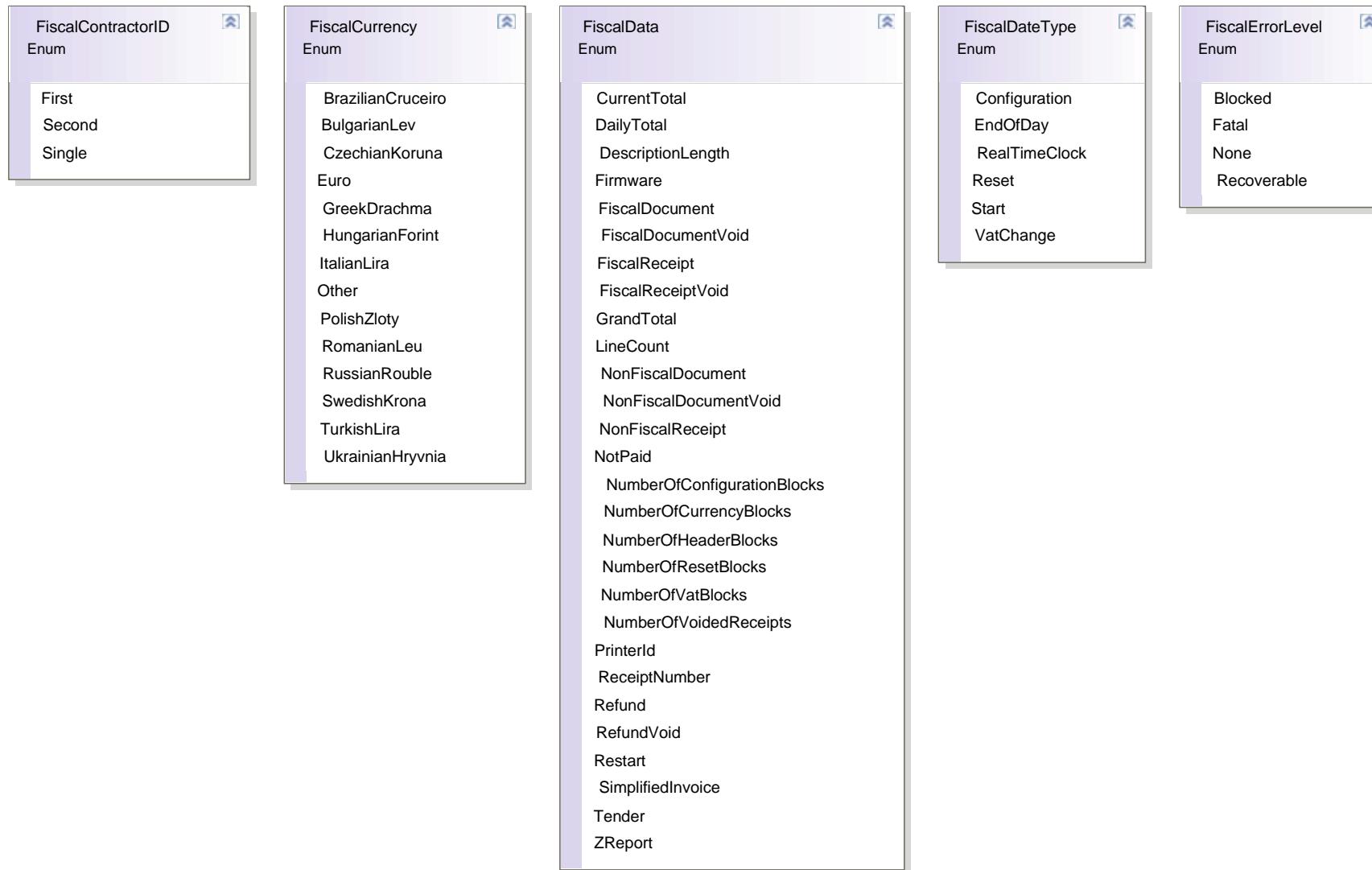
- SetFiscalReceiptStation(FiscalReceiptStation FiscalReceiptStation) : void
- SetFiscalReceiptType(FiscalReceiptType FiscalReceiptType) : void
- SetFlagWhenIdle(bool FlagWhenIdle) : void
- SetFreezeEvents(bool FreezeEvents) : void
- SetHeaderLine(int LineNumber, string Text, bool DoubleWidth) : void
- SetMessageType(FiscalMessageType MessageType) : void
- SetPOSID(string POSID, string CashierID) : void
- SetPostLine(string PostLine) : void
- SetPowerNotify(PowerNotification PowerNotify) : void
- SetPreLine(string PreLine) : void
- SetSlipSelection(FiscalSlipSelection SlipSelection) : void
- SetStoreFiscalID(string ID) : void
- SetTotalizerType(FiscalTotalizerType TotalizerType) : void
- SetTrailerLine(int LineNumber, string Text, bool DoubleWidth) : void
- SetVatTable() : void
- SetVatValue(int VatID, string VatValue) : void
- UpdateFirmware(string FirmwareFileName) : void
- UpdateStatistics(StatisticList StatisticsBuffer) : void
- VerifyItem(string ItemName, int VatID) : void

WS-POS 1.1 Technical Specification

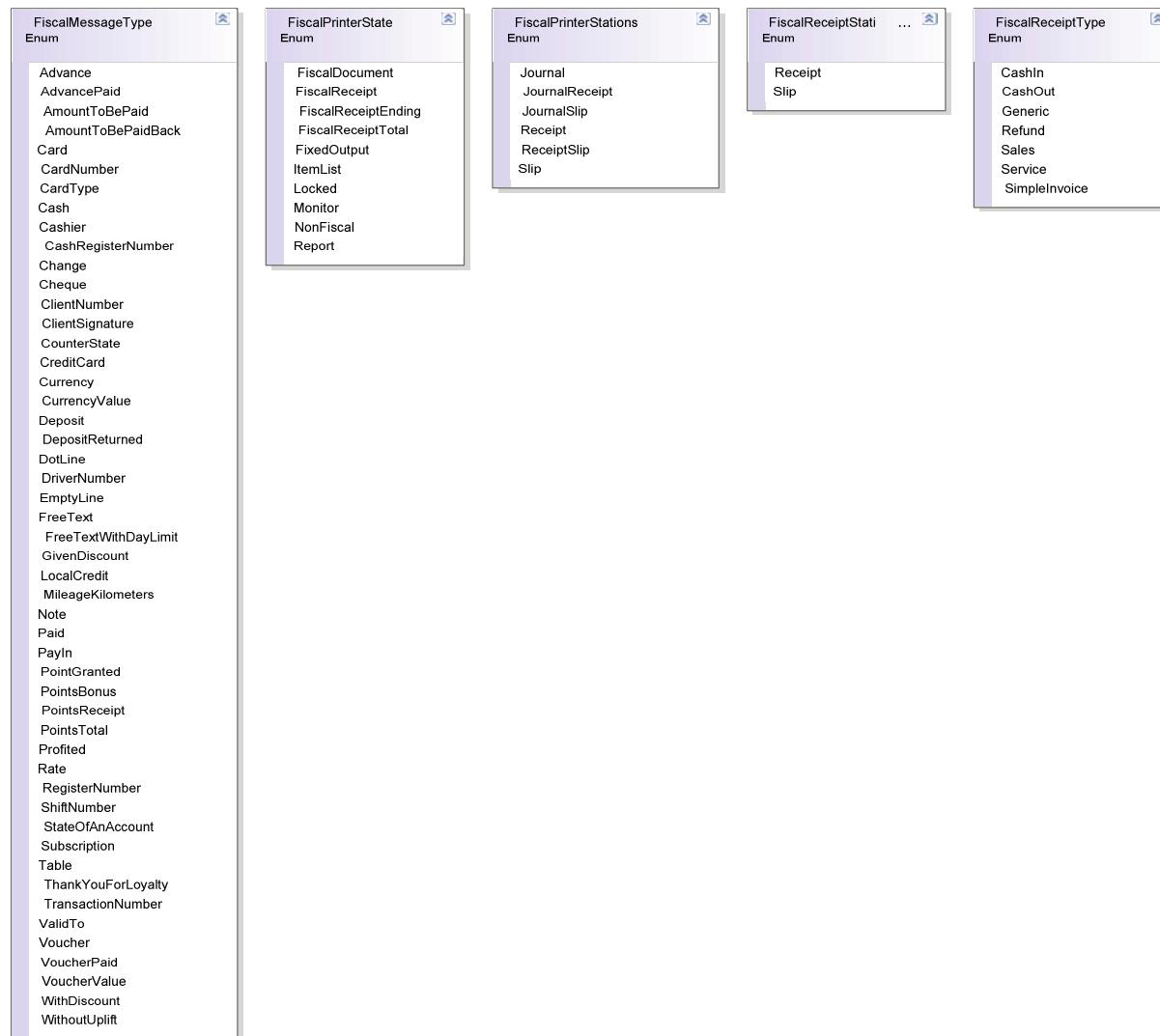
Fiscal Printer



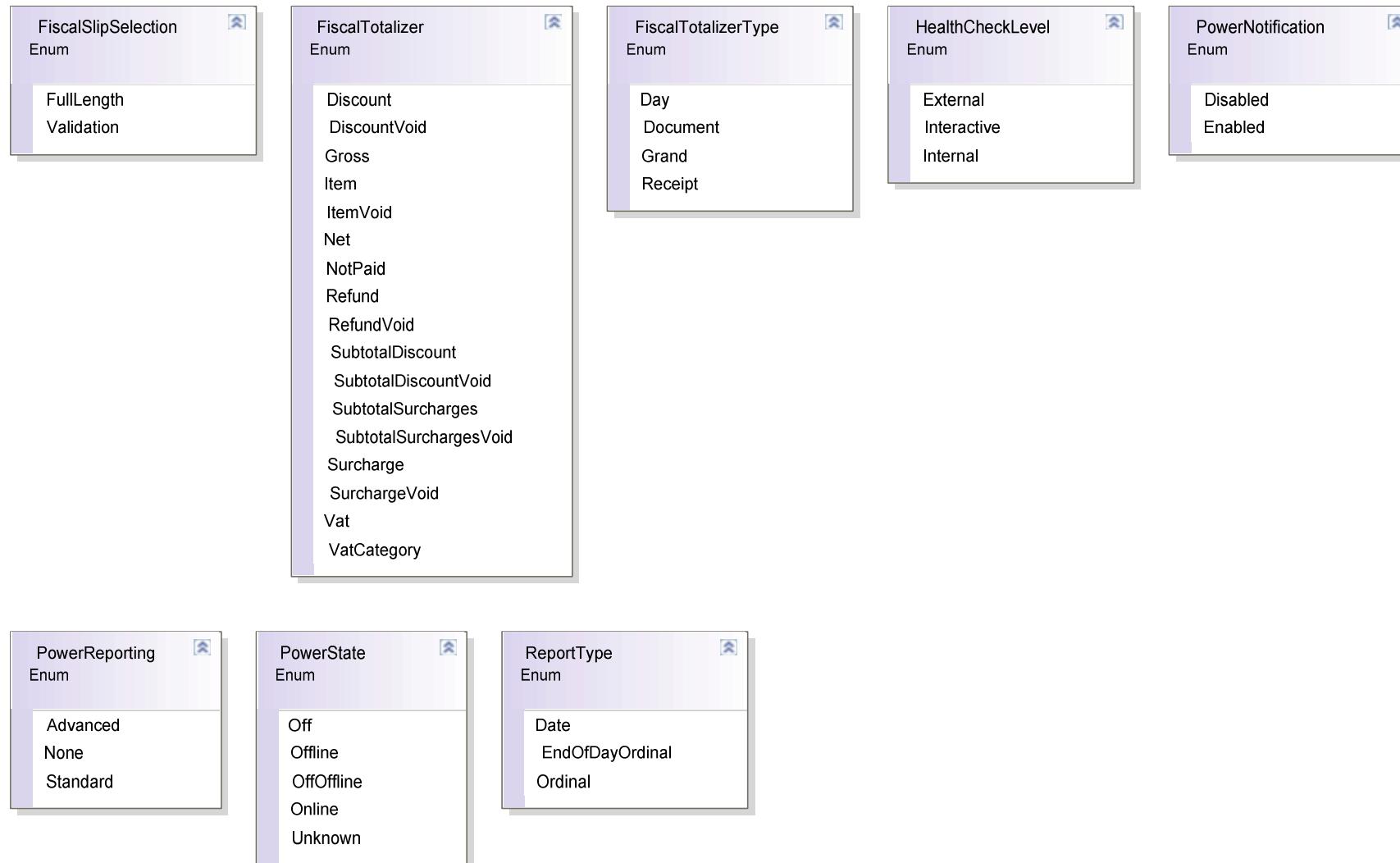
Fiscal Printer



Fiscal Printer

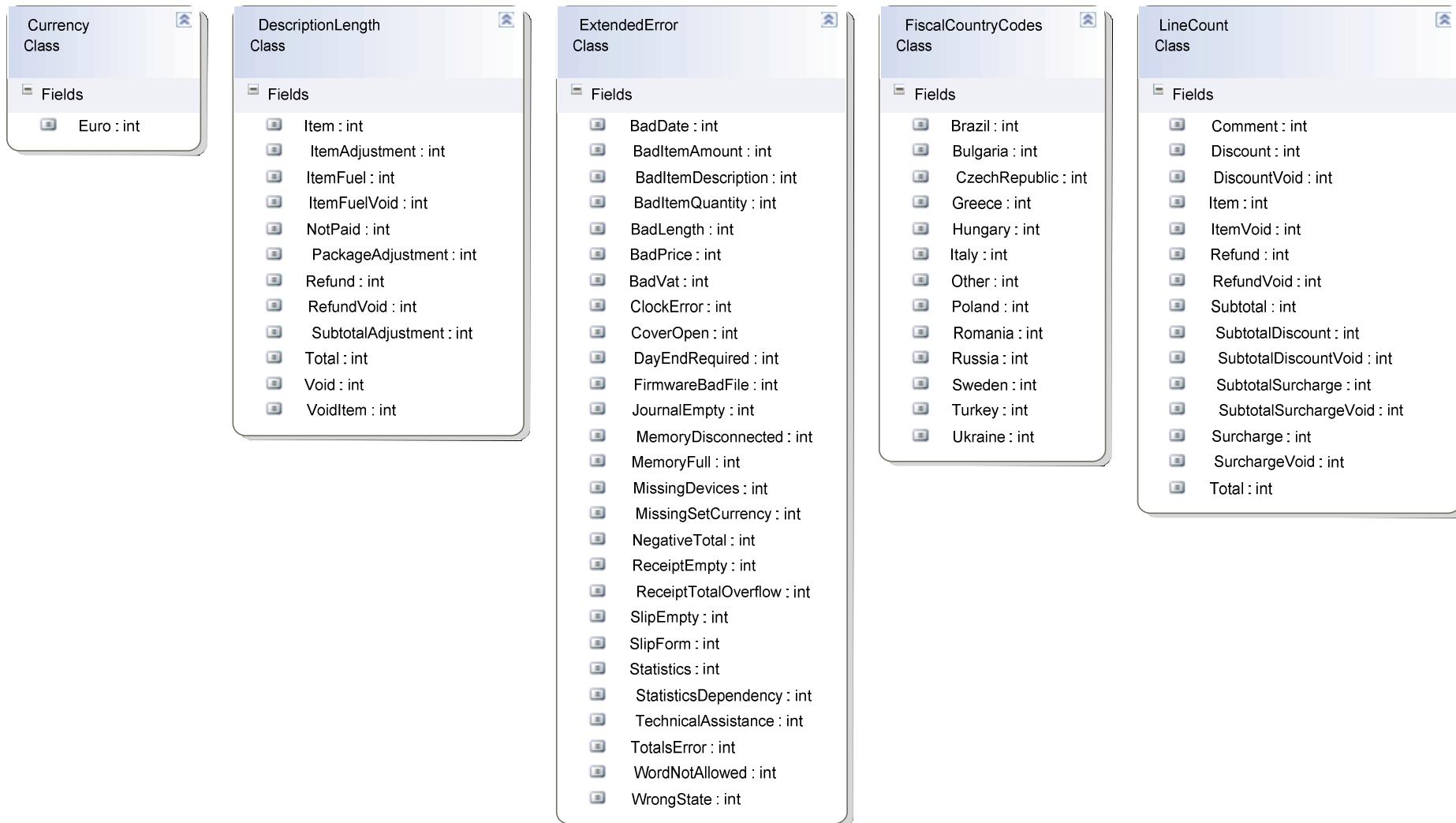


Fiscal Printer

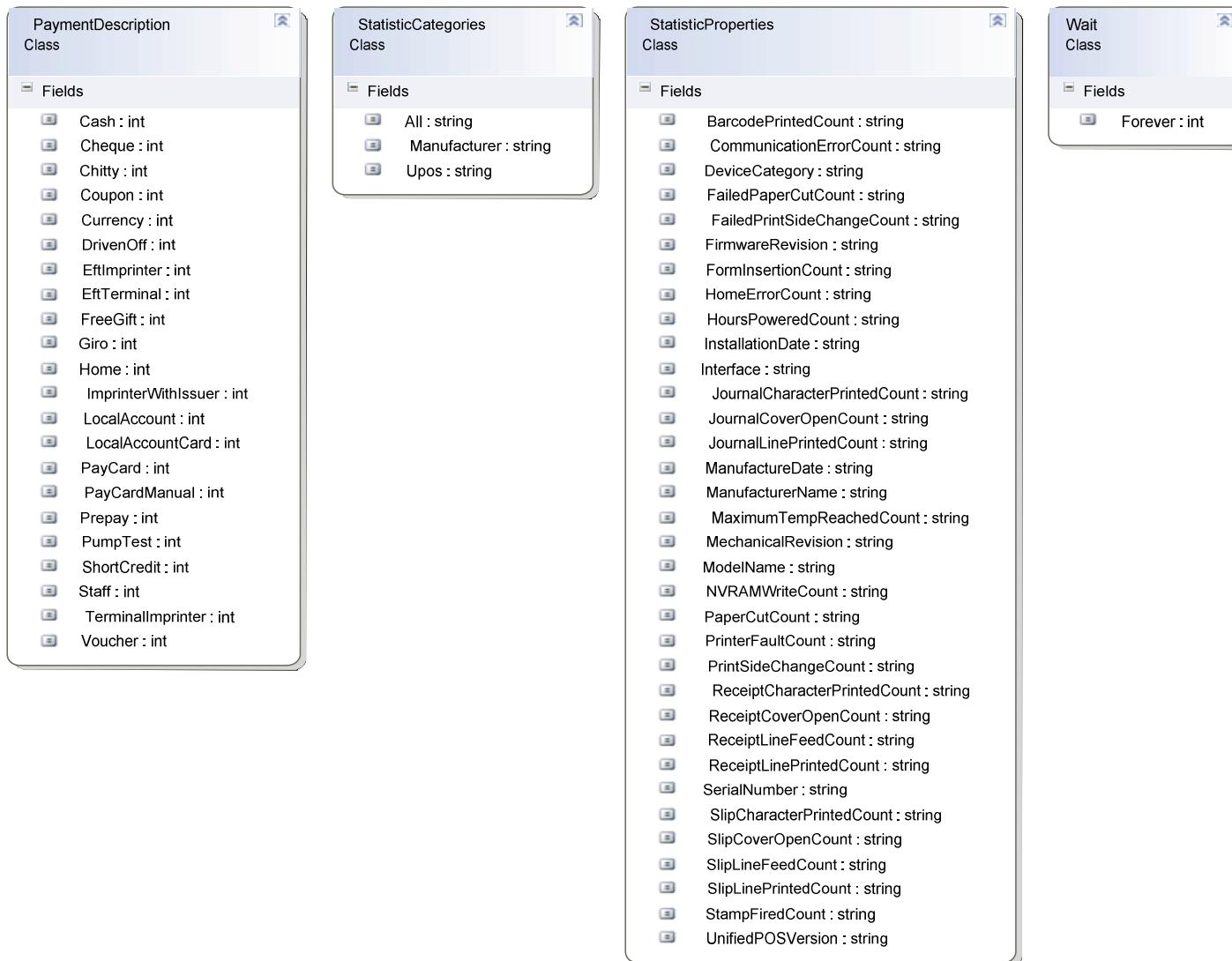


WS-POS 1.1 Technical Specification

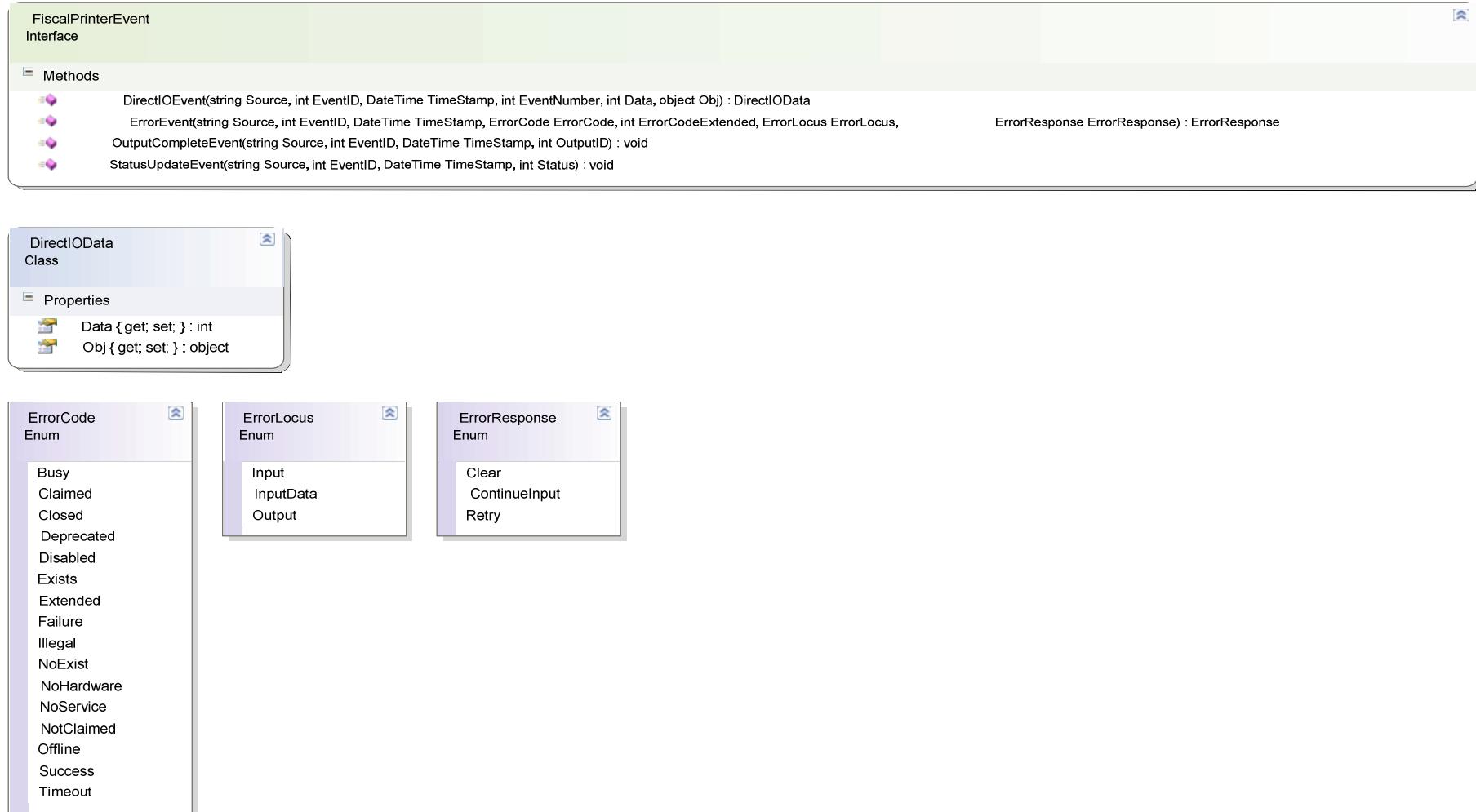
Fiscal Printer



Fiscal Printer



Fiscal Printer Events

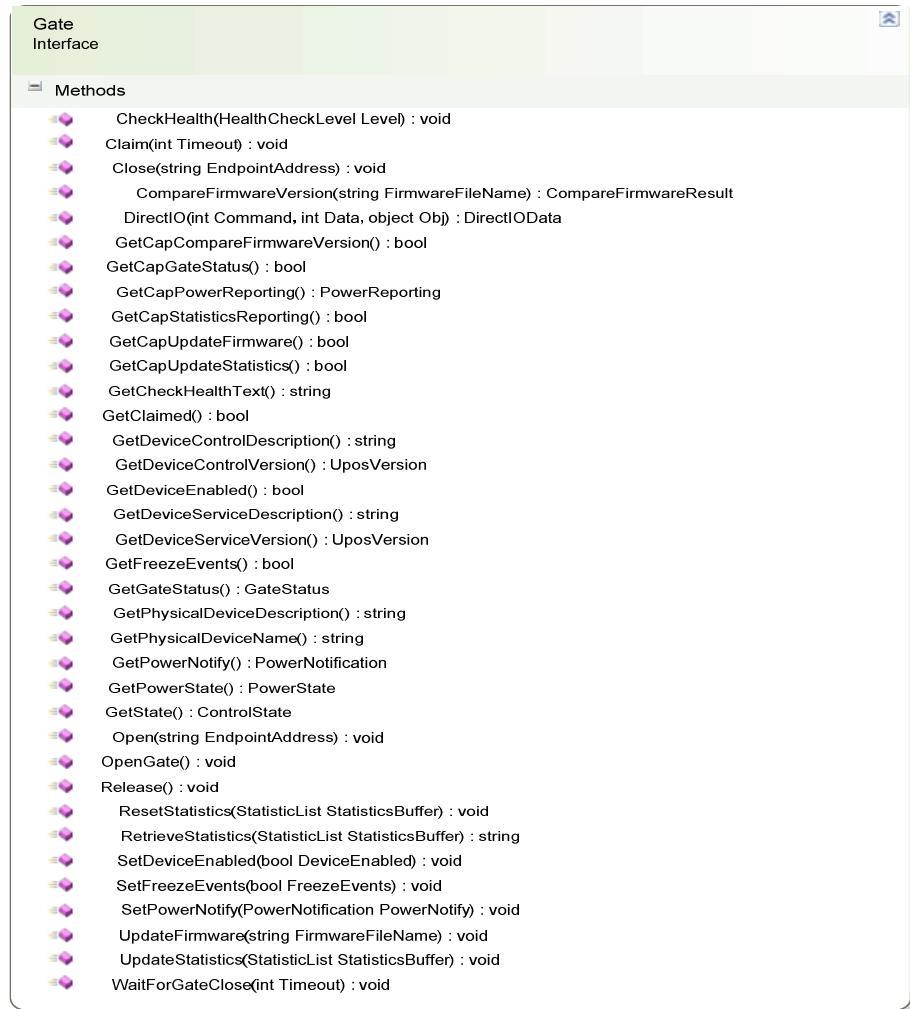


Fiscal Printer Events

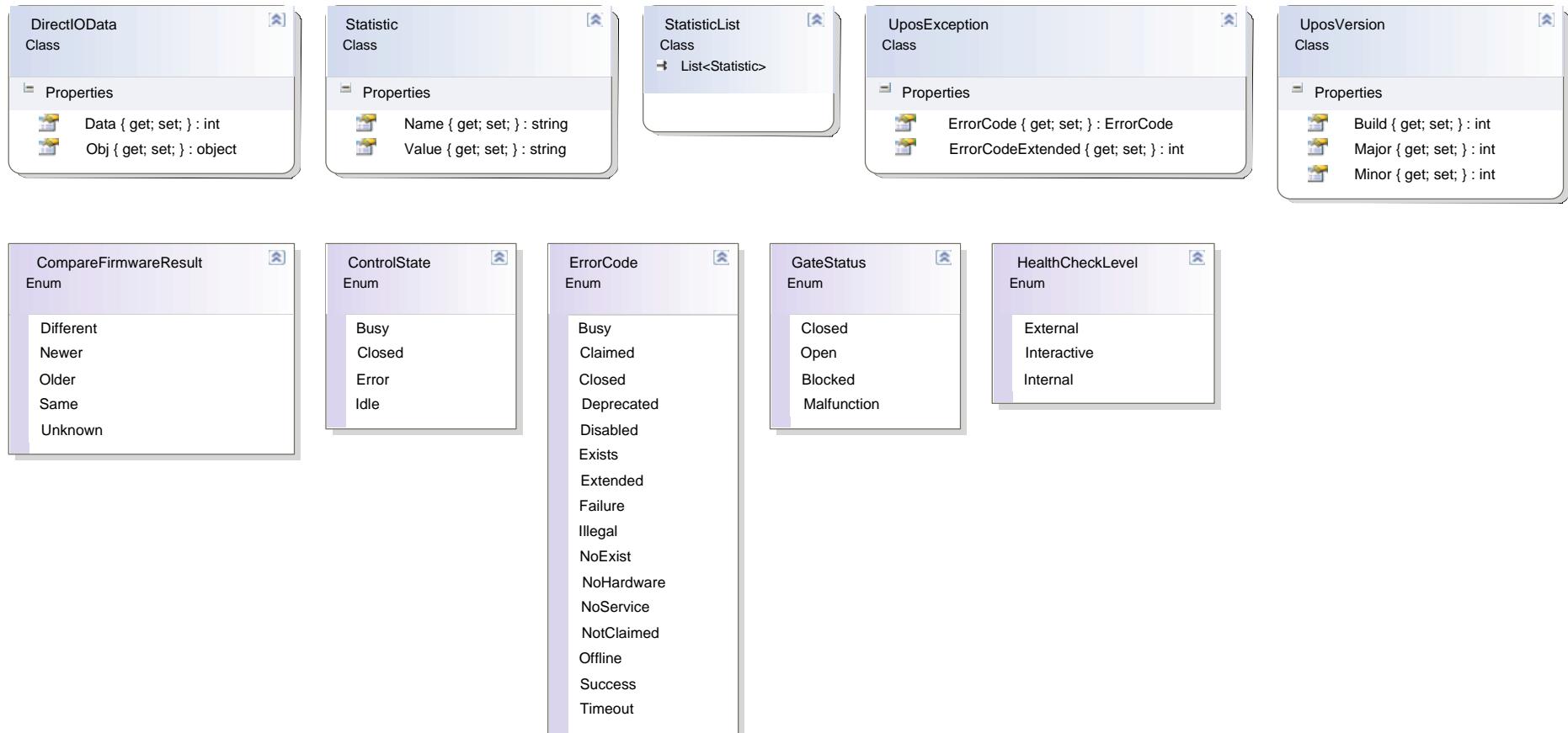
ExtendedError Class	
Fields	
BadDate : int	
BadItemAmount : int	
BadItemDescription : int	
BadItemQuantity : int	
BadLength : int	
BadPrice : int	
BadVat : int	
ClockError : int	
CoverOpen : int	
DayEndRequired : int	
FirmwareBadFile : int	
JournalEmpty : int	
MemoryDisconnected : int	
MemoryFull : int	
MissingDevices : int	
MissingSetCurrency : int	
NegativeTotal : int	
ReceiptEmpty : int	
ReceiptTotalOverflow : int	
SlipEmpty : int	
SlipForm : int	
Statistics : int	
StatisticsDependency : int	
TechnicalAssistance : int	
TotalsError : int	
WordNotAllowed : int	
WrongState : int	

Status Class	
Fields	
CoverOK : int	
CoverOpen : int	
Idle : int	
JournalCoverOK : int	
JournalCoverOpen : int	
JournalEmpty : int	
JournalNearEmpty : int	
JournalPaperOK : int	
PowerOff : int	
PowerOffline : int	
PowerOffOffline : int	
PowerOnline : int	
ReceiptCoverOK : int	
ReceiptCoverOpen : int	
ReceiptEmpty : int	
ReceiptNearEmpty : int	
ReceiptPaperOK : int	
SlipCoverOK : int	
SlipCoverOpen : int	
SlipEmpty : int	
SlipNearEmpty : int	
SlipPaperOK : int	
UpdateFirmwareComplete : int	
UpdateFirmwareCompleteDeviceNotRestored : int	
UpdateFirmwareFailedDeviceNeedsFirmware : int	
UpdateFirmwareFailedDeviceOk : int	
UpdateFirmwareFailedDeviceUnknown : int	
UpdateFirmwareFailedDeviceUnrecoverable : int	
UpdateFirmwareProgress : int	

Gate

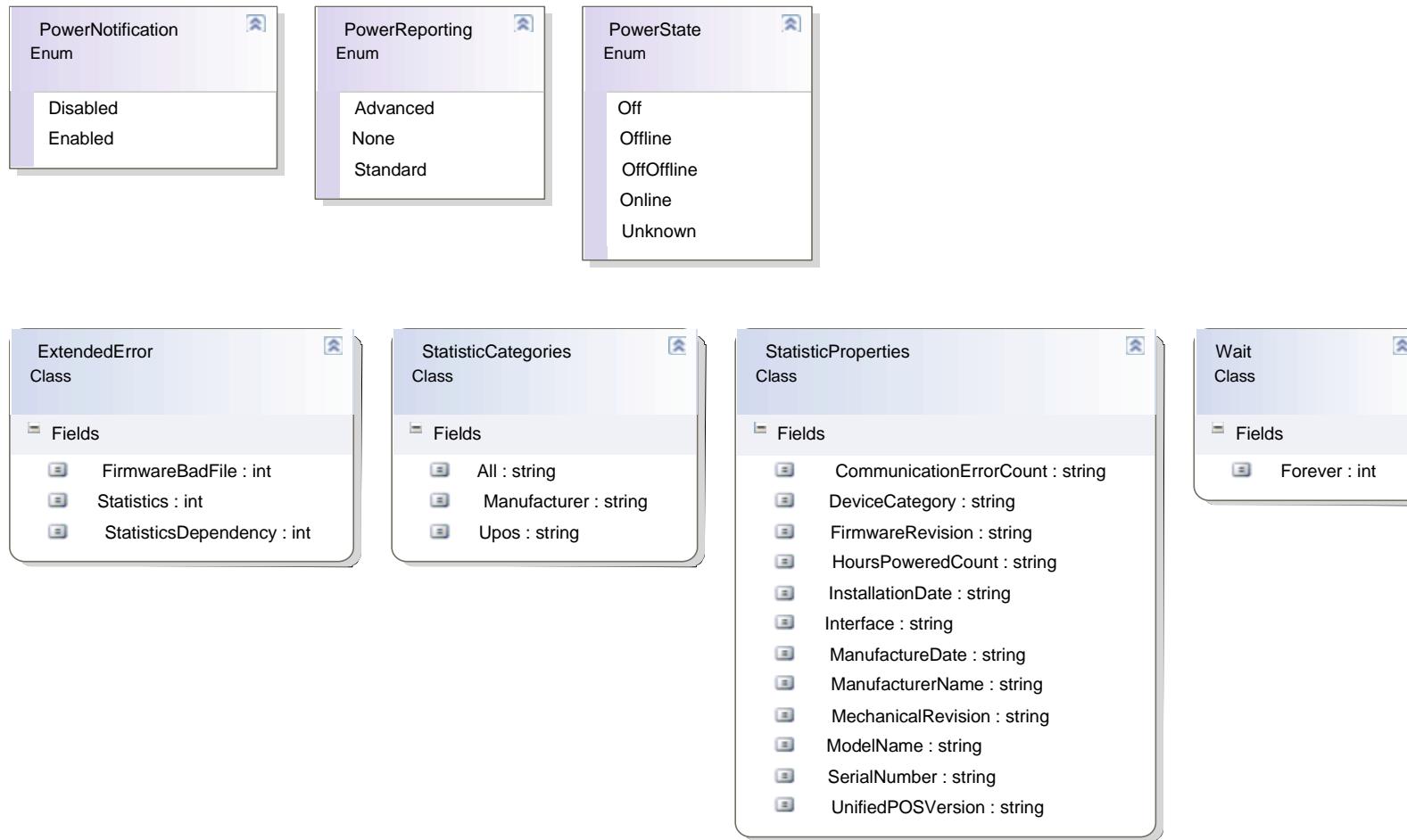


Gate

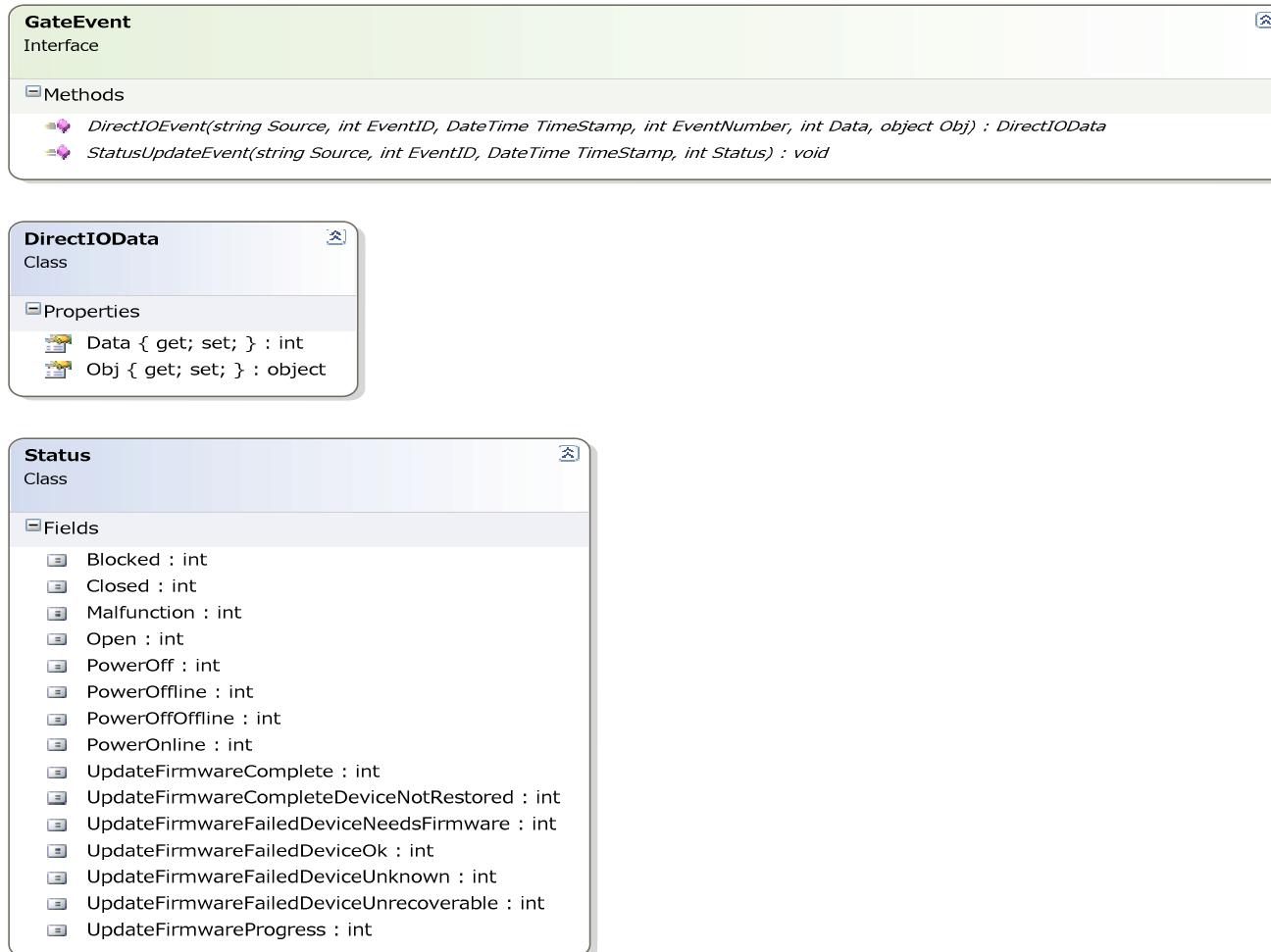


WS-POS 1.1 Technical Specification

Gate



Gate Events



Hard Totals

HardTotals Interface	
Methods	
 BeginTrans() : void	
 CheckHealth(HealthCheckLevel Level) : void	
 Claim(int Timeout) : void	
 ClaimFile(int Handle, int Timeout) : void	
 Close(string EndpointAddress) : void	
 CommitTrans() : void	
 CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult	
 Create(string FileName, int Size, bool ErrorDetection) : int	
 Delete(string FileName) : void	
 DirectIO(int Command, int Data, object Obj) : DirectIOData	
 Find(string FileName) : TotalsFileInfo	
 FindByIndex(int Index) : string	
 GetCapCompareFirmwareVersion() : bool	
 GetCapErrorDetection() : bool	
 GetCapPowerReporting() : PowerReporting	
 GetCapSingleFile() : bool	
 GetCapStatisticsReporting() : bool	
 GetCapTransactions() : bool	
 GetCapUpdateFirmware() : bool	
 GetCapUpdateStatistics() : bool	
 GetCheckHealthText() : string	
 GetClaimed() : bool	
 GetDeviceControlDescription() : string	
 GetDeviceControlVersion() : UposVersion	
 GetDeviceEnabled() : bool	
 GetDeviceServiceDescription() : string	
 GetDeviceServiceVersion() : UposVersion	

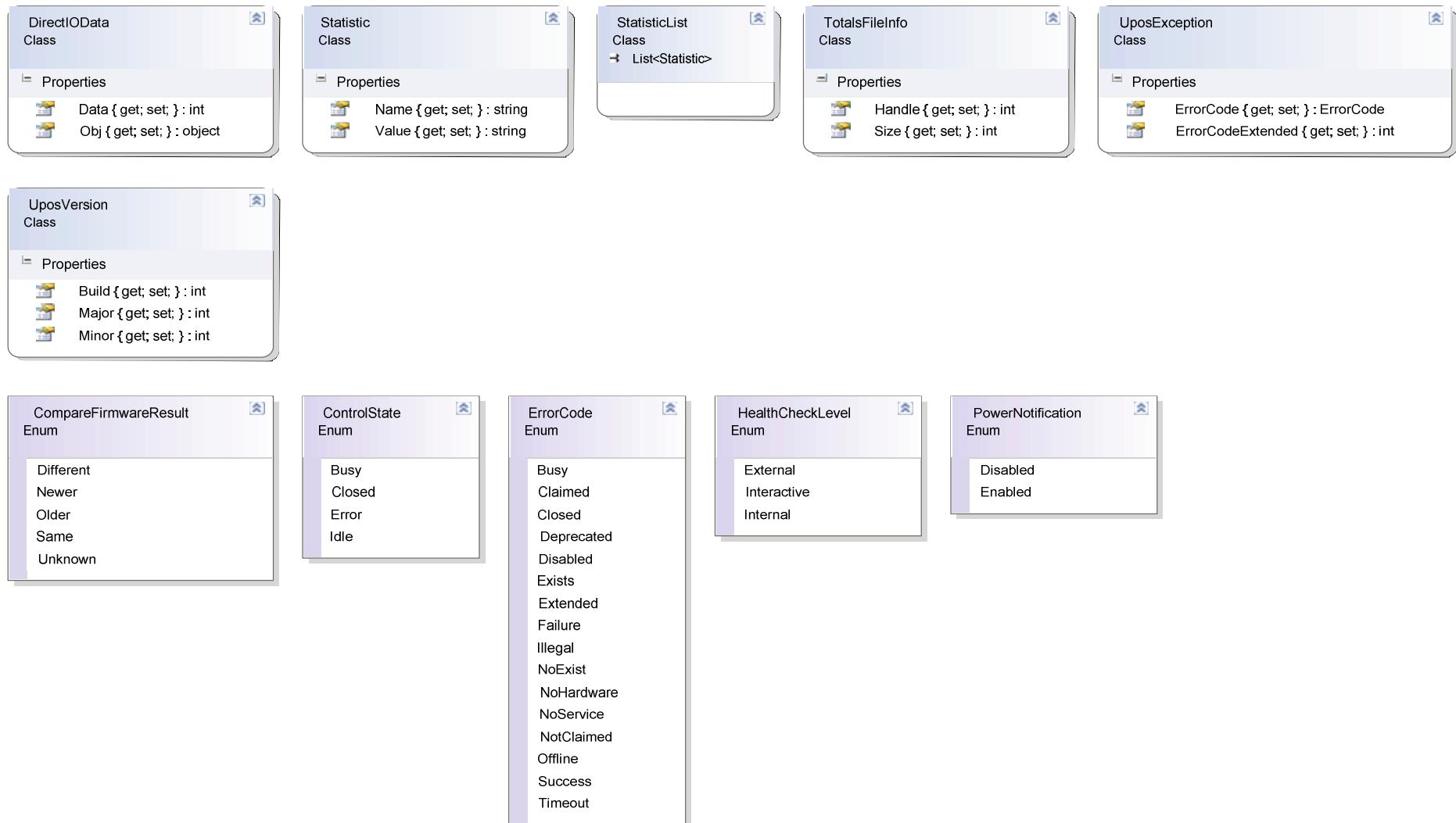
Hard Totals



- GetFreeData() : int
- GetFreezeEvents() : bool
- GetNumberOfFiles() : int
- GetPhysicalDeviceDescription() : string
- GetPhysicalDeviceName() : string
- GetPowerNotify() : PowerNotification
- GetPowerState() : PowerState
- GetState() : ControlState
- GetTotalsSize() : int
- GetTransactionInProgress() : bool
- Open(string EndpointAddress) : void
- Read(int Handle, int Offset, int Count) : byte[]
- RecalculateValidationData(int Handle) : void
- Release() : void
- ReleaseFile(int Handle) : void
- Rename(int Handle, string NewName) : void
- ResetStatistics(StatisticList StatisticsBuffer) : void
- RetrieveStatistics(StatisticList StatisticsBuffer) : string
- Rollback() : void
- SetAll(int Handle, sbyte TargetValue) : void
- SetDeviceEnabled(bool DeviceEnabled) : void
- SetFreezeEvents(bool FreezeEvents) : void
- SetPowerNotify(PowerNotification PowerNotify) : void
- UpdateFirmware(string FirmwareFileName) : void
- UpdateStatistics(StatisticList StatisticsBuffer) : void
- ValidateData(int Handle) : void
- Write(int Handle, byte[] Data, int Offset) : void

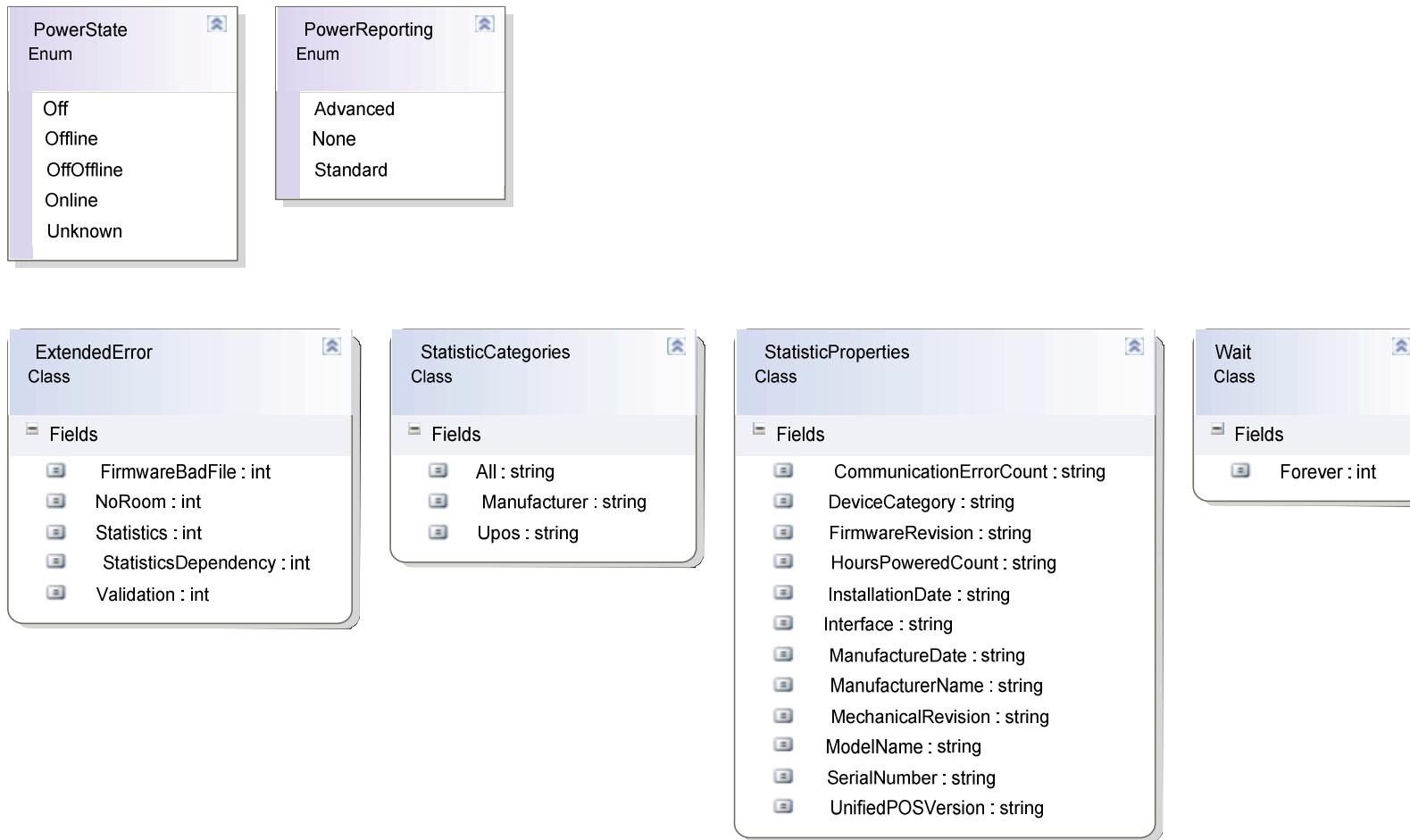
WS-POS 1.1 Technical Specification

Hard Totals



WS-POS 1.1 Technical Specification

Hard Totals



Hard Totals Events

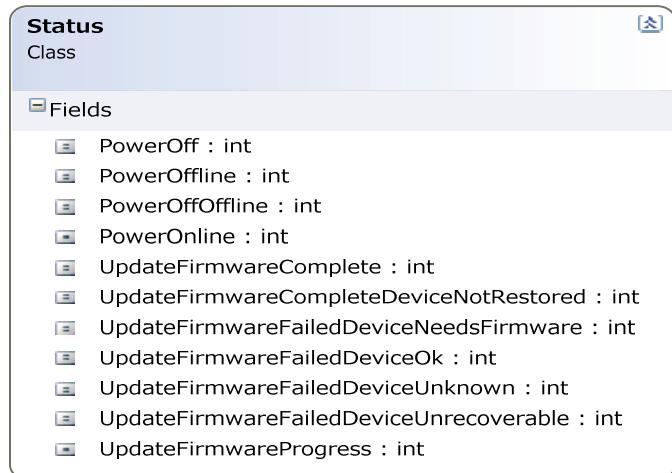
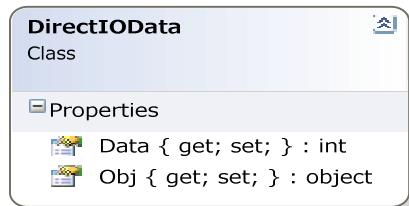


Image Scanner

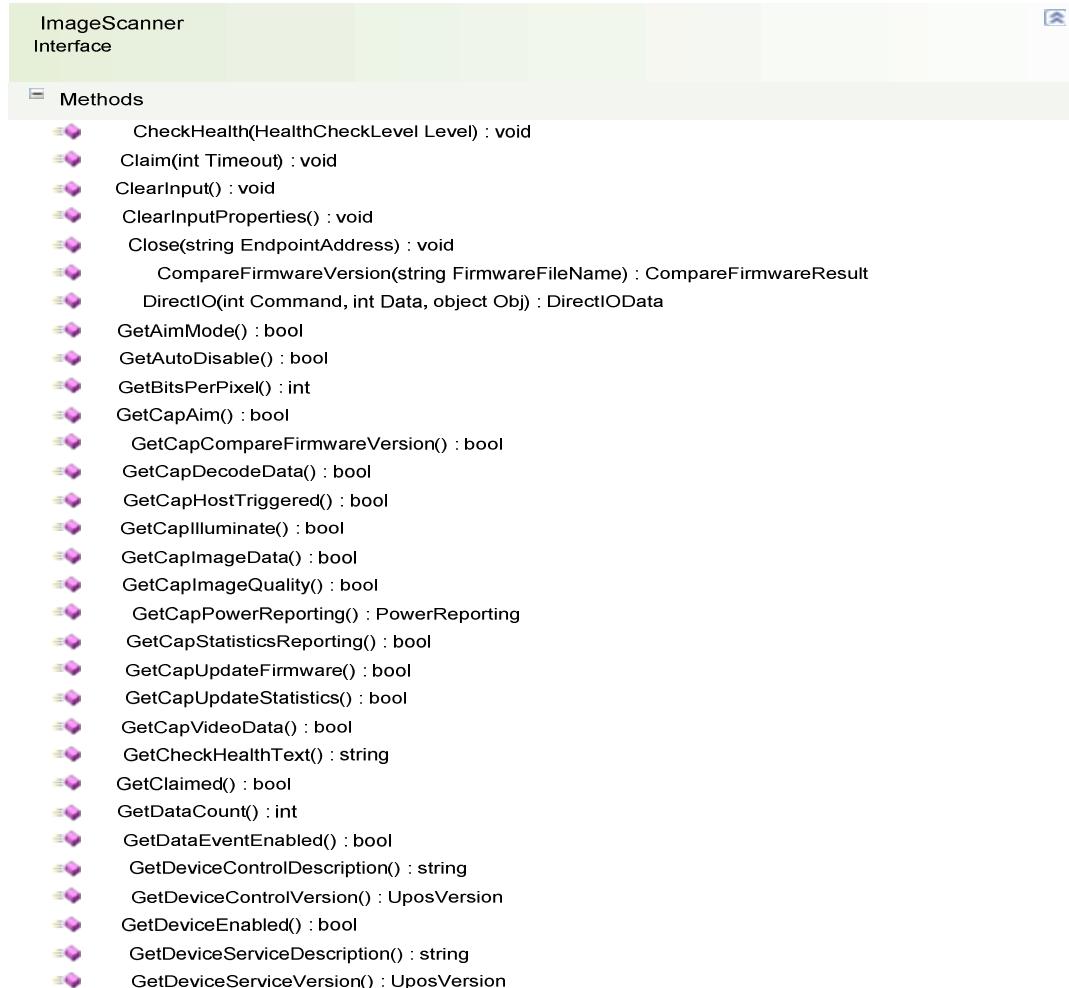
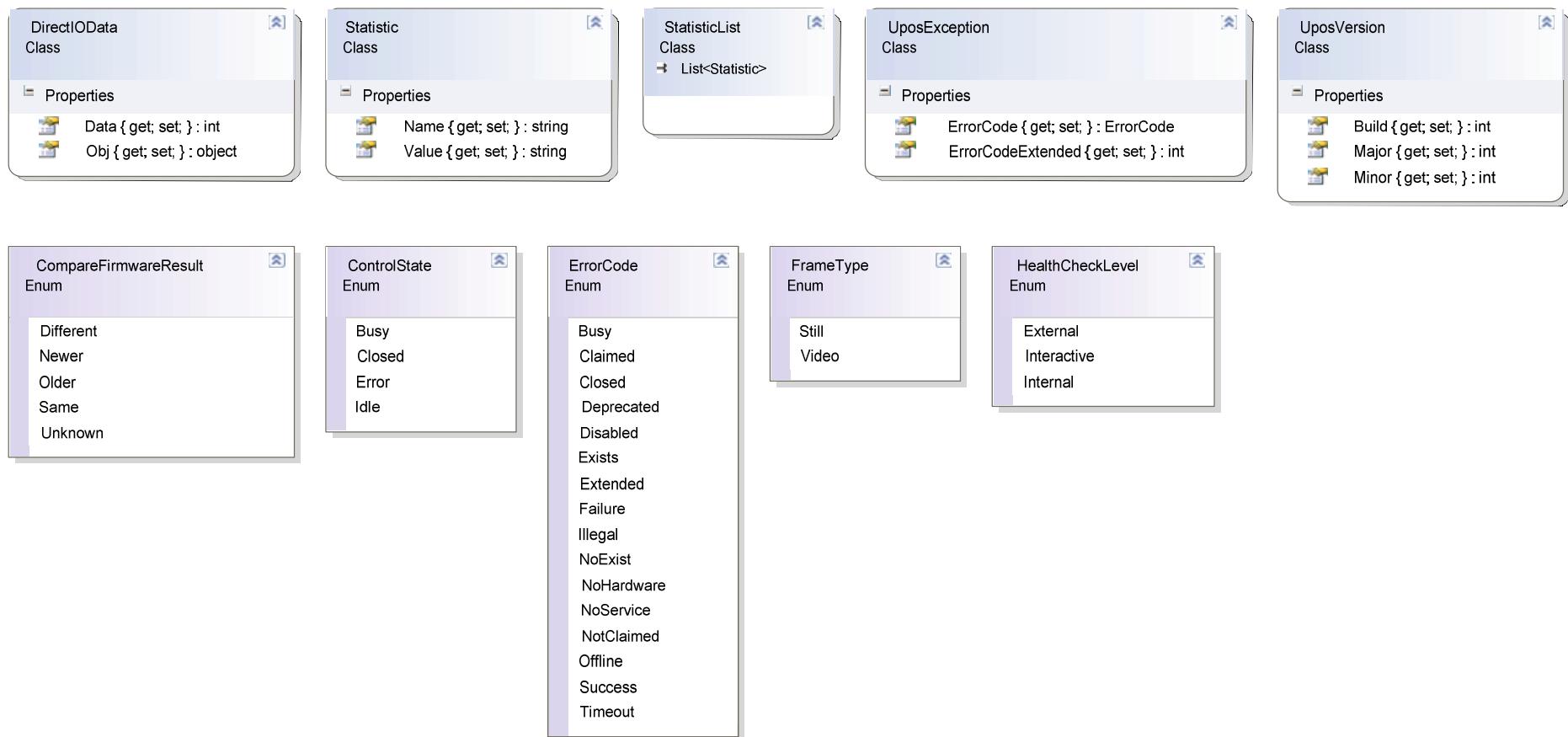


Image Scanner



WS-POS 1.1 Technical Specification

Image Scanner



WS-POS 1.1 Technical Specification

Image Scanner

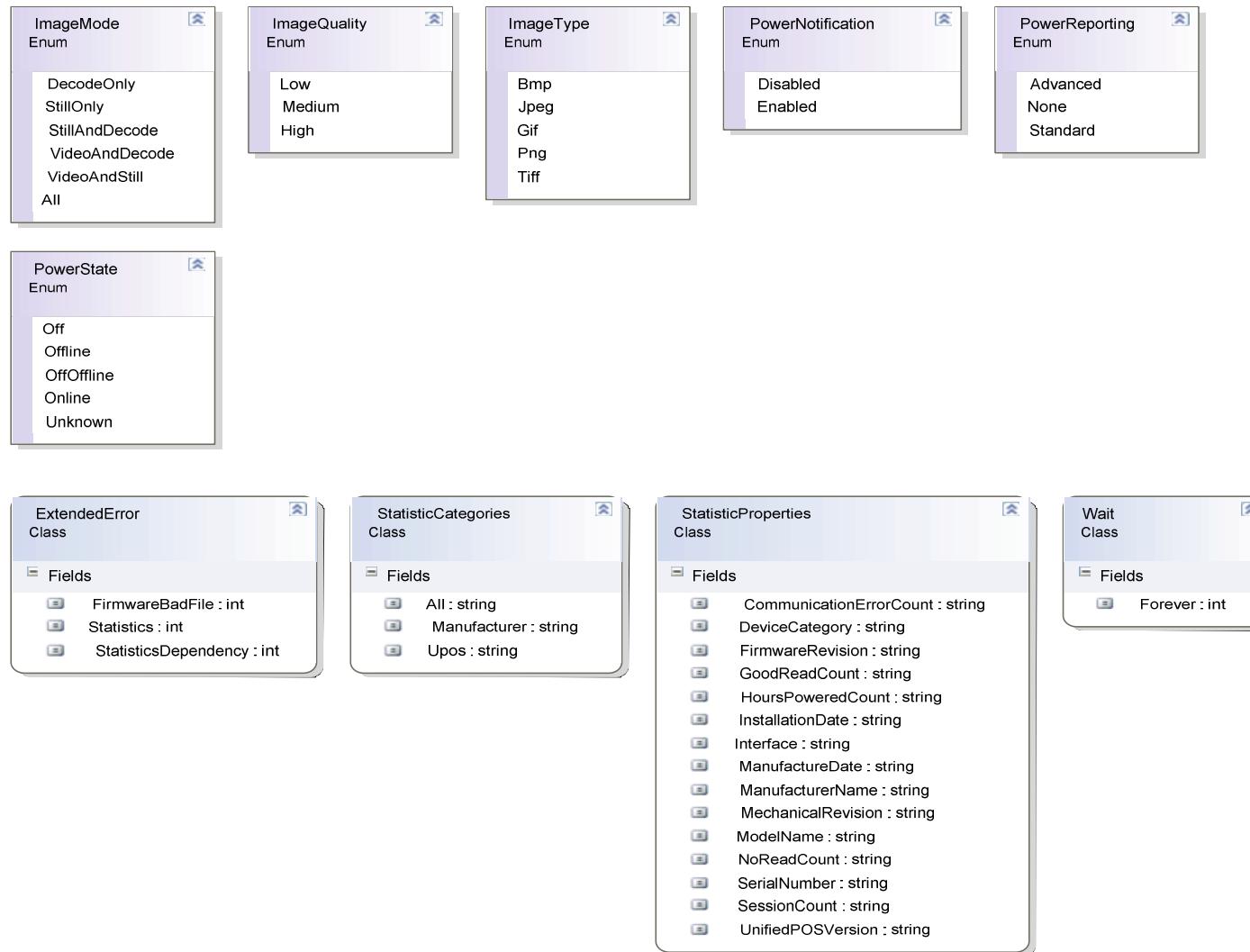


Image Scanner Events



Image Scanner Events



Item Dispenser

ItemDispenser
Interface

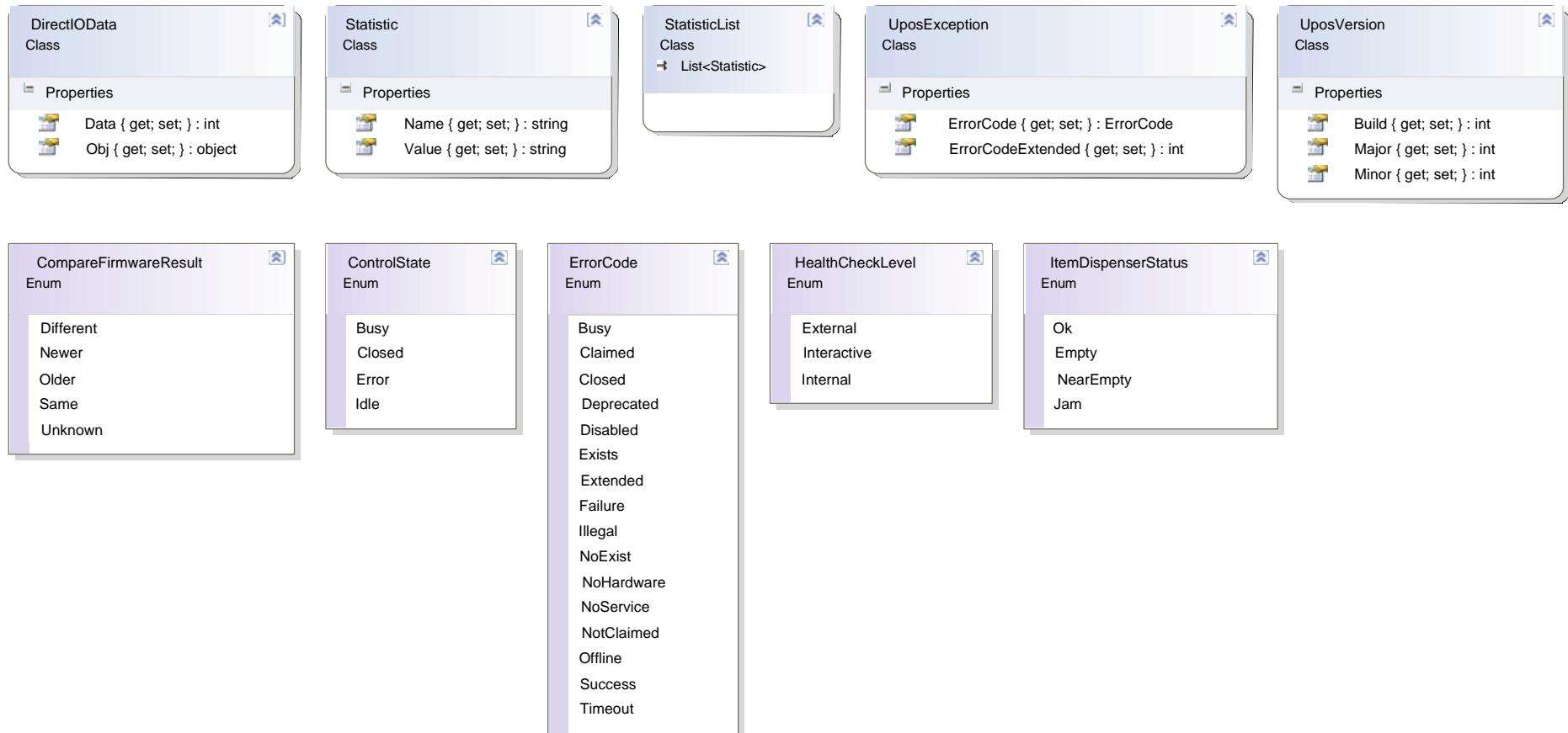
Methods

- AdjustItemCount(int ItemCount, int SlotNumber) : void
- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- Close(string EndpointAddress) : void
- CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
- DirectIO(int Command, int Data, object Obj) : DirectIOData
- DispenseItem(int NumItem, int SlotNumber) : int
- GetCapCompareFirmwareVersion() : bool
- GetCapEmptySensor() : bool
- GetCapIndividualSlotStatus() : bool
- GetCapJamSensor() : bool
- GetCapNearEmptySensor() : bool
- GetCapPowerReporting() : PowerReporting
- GetCapStatisticsReporting() : bool
- GetCapUpdateFirmware() : bool
- GetCapUpdateStatistics() : bool
- GetCheckHealthText() : string
- GetClaimed() : bool
- GetDeviceControlDescription() : string
- GetDeviceControlVersion() : UposVersion
- GetDeviceEnabled() : bool
- GetDeviceServiceDescription() : string

Item Dispenser

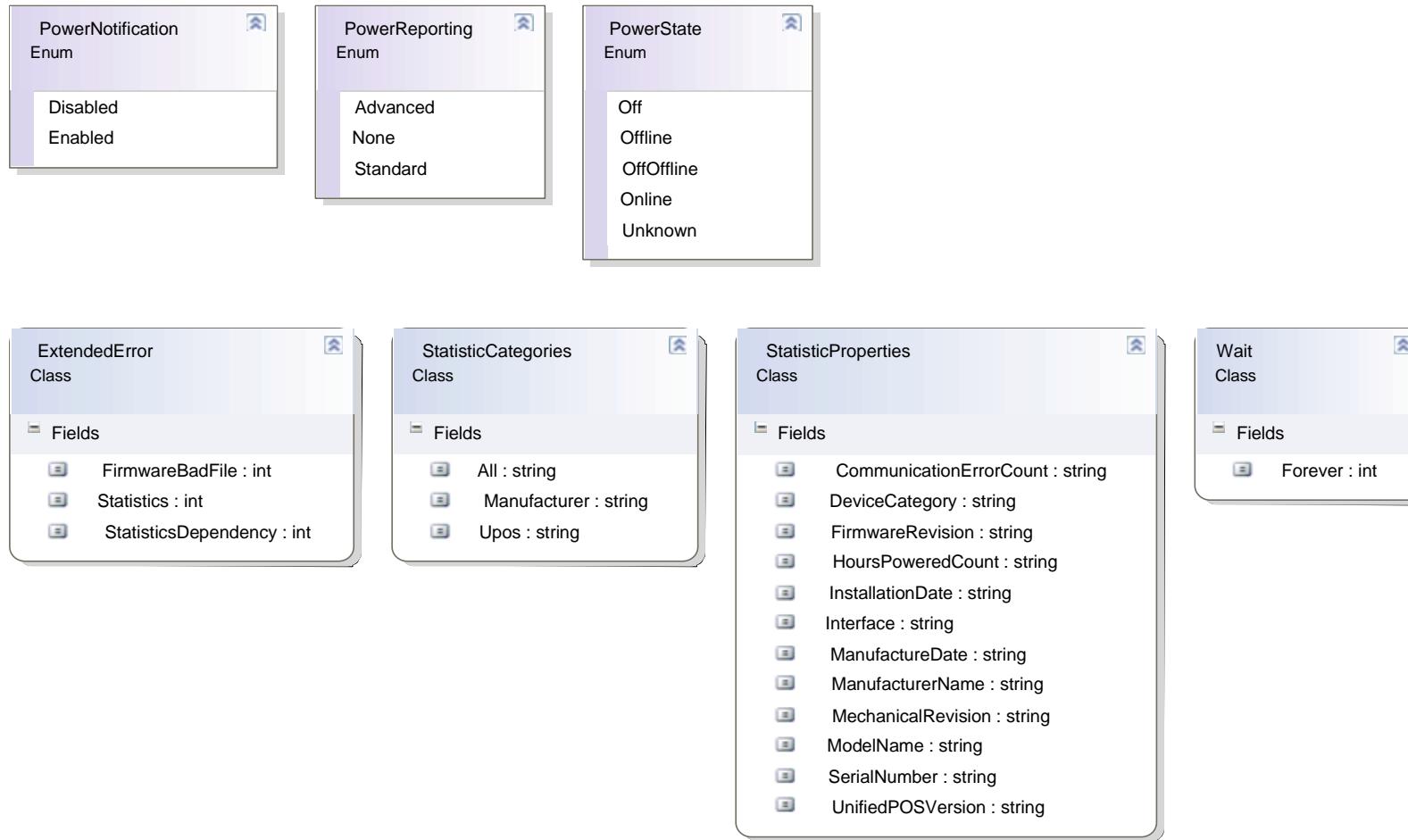


Item Dispenser

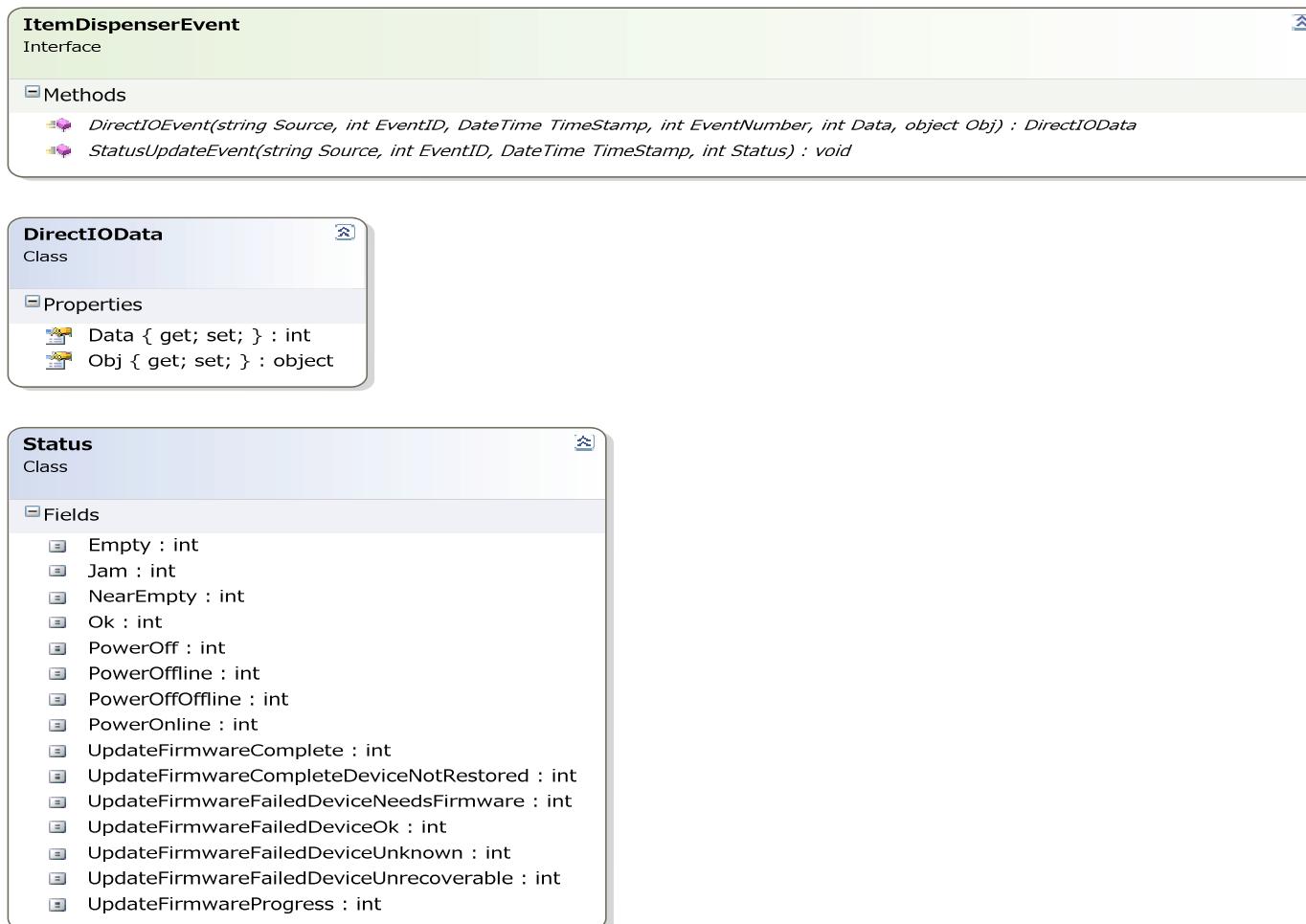


WS-POS 1.1 Technical Specification

Item Dispenser



Item Dispenser Events



Keylock

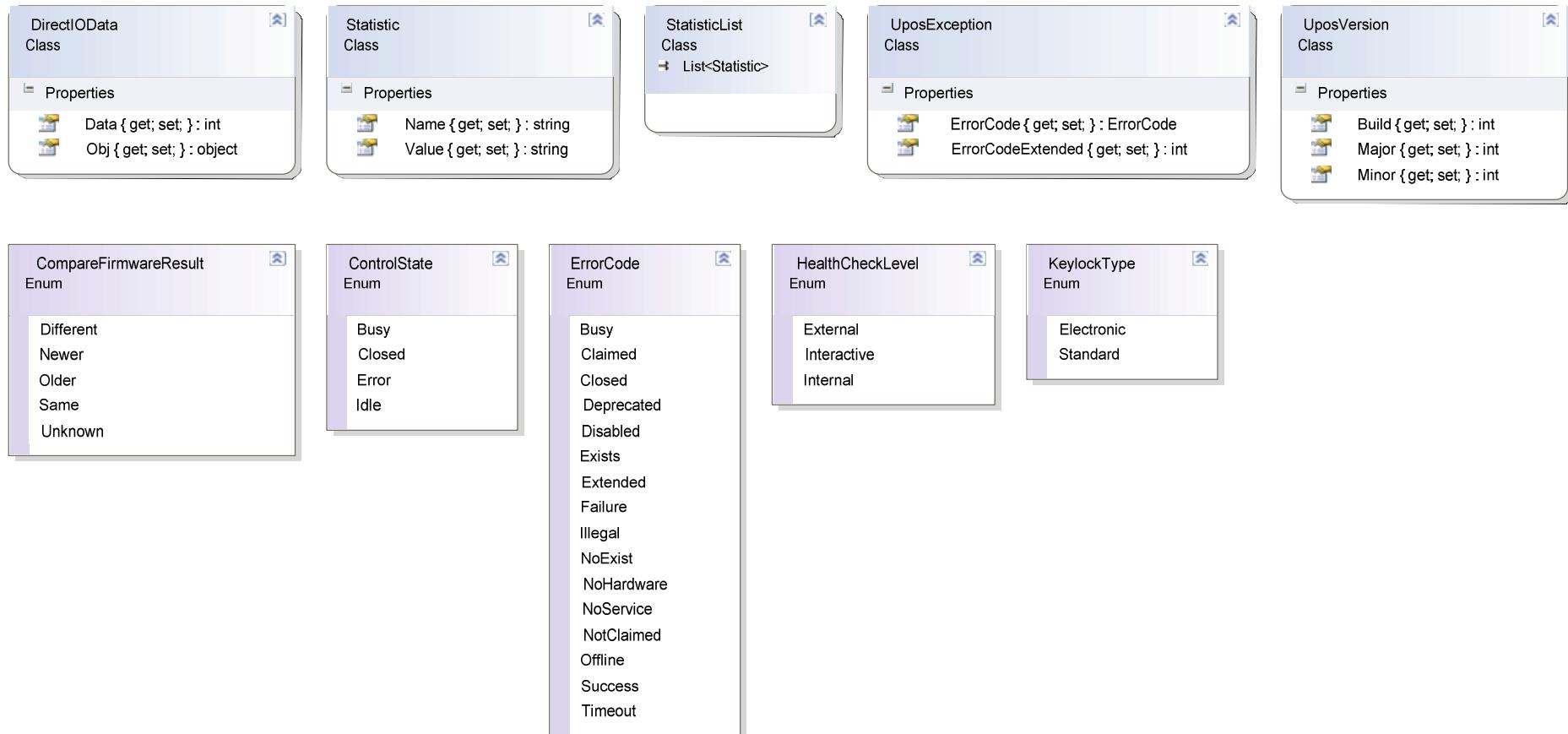
The diagram shows a UML class named "Keylock Interface". It has a single section labeled "Methods" which contains the following list of methods:

- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- Close(string EndpointAddress) : void
- CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
- DirectIO(int Command, int Data, object Obj) : DirectIOData
- GetCapCompareFirmwareVersion() : bool
- GetCapKeylockType() : KeylockType
- GetCapPowerReporting() : PowerReporting
- GetCapStatisticsReporting() : bool
- GetCapUpdateFirmware() : bool
- GetCapUpdateStatistics() : bool
- GetCheckHealthText() : string
- GetClaimed() : bool
- GetDeviceControlDescription() : string
- GetDeviceControlVersion() : UposVersion
- GetDeviceEnabled() : bool
- GetDeviceServiceDescription() : string
- GetDeviceServiceVersion() : UposVersion
- GetElectronicKeyValue() : byte[]

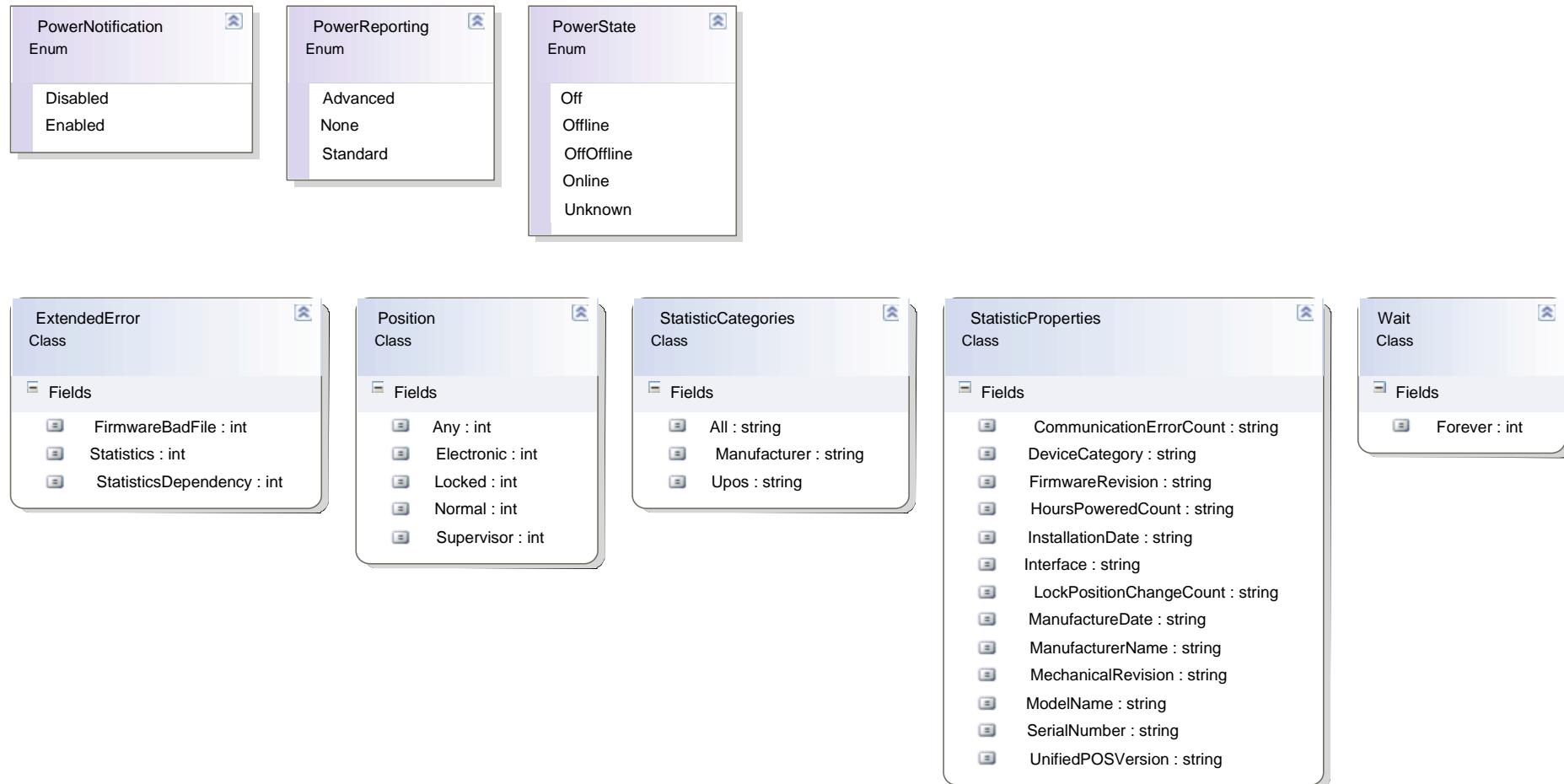
Keylock

- ≡ GetFreezeEvents() : bool
- ≡ GetKeyPosition() : int
- ≡ GetPhysicalDeviceDescription() : string
- ≡ GetPhysicalDeviceName() : string
- ≡ GetPositionCount() : int
- ≡ GetPowerNotify() : PowerNotification
- ≡ GetPowerState() : PowerState
- ≡ GetState() : ControlState
- ≡ Open(string EndpointAddress) : void
- ≡ Release() : void
- ≡ ResetStatistics(StatisticList StatisticsBuffer) : void
- ≡ RetrieveStatistics(StatisticList StatisticsBuffer) : string
- ≡ SetDeviceEnabled(bool DeviceEnabled) : void
- ≡ SetFreezeEvents(bool FreezeEvents) : void
- ≡ SetPowerNotify(PowerNotification PowerNotify) : void
- ≡ UpdateFirmware(string FirmwareFileName) : void
- ≡ UpdateStatistics(StatisticList StatisticsBuffer) : void
- ≡ WaitForKeylockChange(int KeyPosition, int Timeout) : void

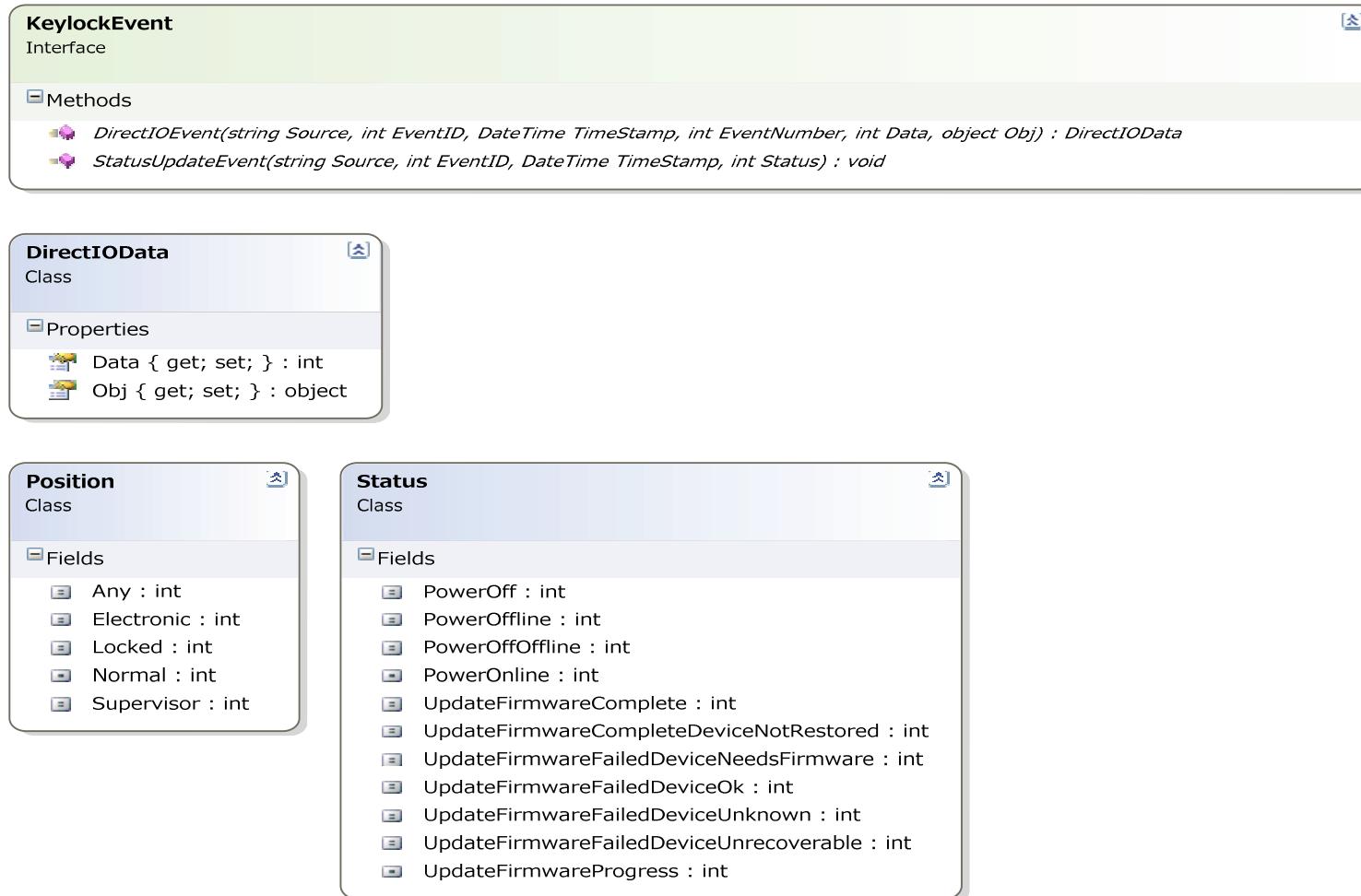
Keylock



Keylock



Keylock Events



Lights

The screenshot shows a software interface with a light green header bar containing the text "Lights Interface". Below this is a white main area. On the left side of the main area, there is a tree view with a single expanded node labeled "Methods". This node has 21 children, each represented by a small purple icon followed by a method name and its return type. The methods listed are:

- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- Close(string EndpointAddress) : void
- CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
- DirectIO(int Command, int Data, object Obj) : DirectIOData
- GetCapAlarm() : int
- GetCapBlink() : bool
- GetCapColor() : int
- GetCapCompareFirmwareVersion() : bool
- GetCapPowerReporting() : PowerReporting
- GetCapStatisticsReporting() : bool
- GetCapUpdateFirmware() : bool
- GetCapUpdateStatistics() : bool
- GetCheckHealthText() : string
- GetClaimed() : bool
- GetDeviceControlDescription() : string
- GetDeviceControlVersion() : UposVersion
- GetDeviceEnabled() : bool
- GetDeviceServiceDescription() : string
- GetDeviceServiceVersion() : UposVersion
- GetFreezeEvents() : bool

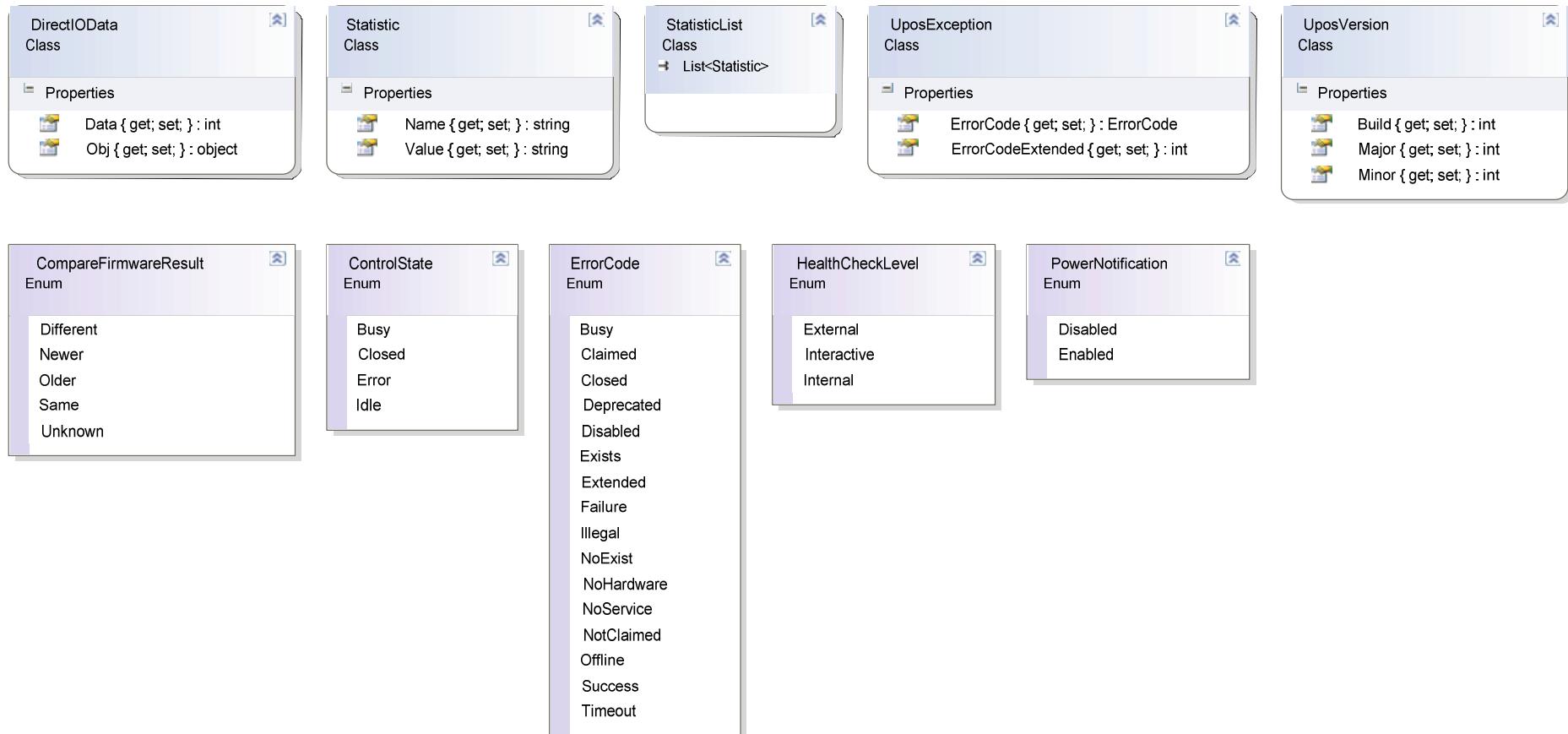
Lights



Methods

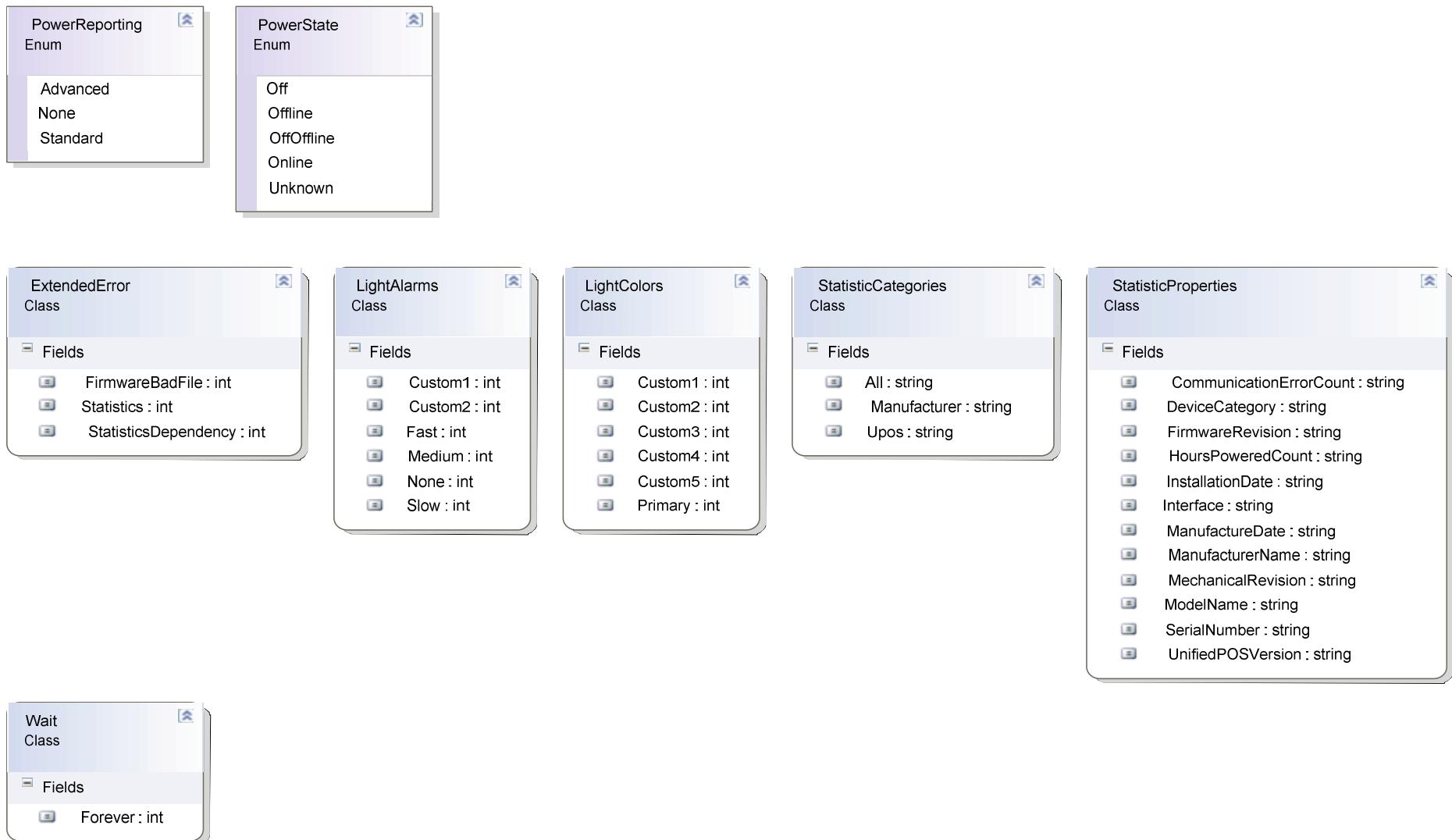
- GetMaxLights() : int
- GetPhysicalDeviceDescription() : string
- GetPhysicalDeviceName() : string
- GetPowerNotify() : PowerNotification
- GetPowerState() : PowerState
- GetState() : ControlState
- Open(string EndpointAddress) : void
- Release() : void
- ResetStatistics(StatisticList StatisticsBuffer) : void
- RetrieveStatistics(StatisticList StatisticsBuffer) : string
- SetDeviceEnabled(bool DeviceEnabled) : void
- SetFreezeEvents(bool FreezeEvents) : void
- SetPowerNotify(PowerNotification PowerNotify) : void
- SwitchOff(int LightNumber) : void
 - SwitchOn(int LightNumber, int BlinkOnCycle, int BlinkOffCycle, int Color, int Alarm) : void
- UpdateFirmware(string FirmwareFileName) : void
- UpdateStatistics(StatisticList StatisticsBuffer) : void

Lights

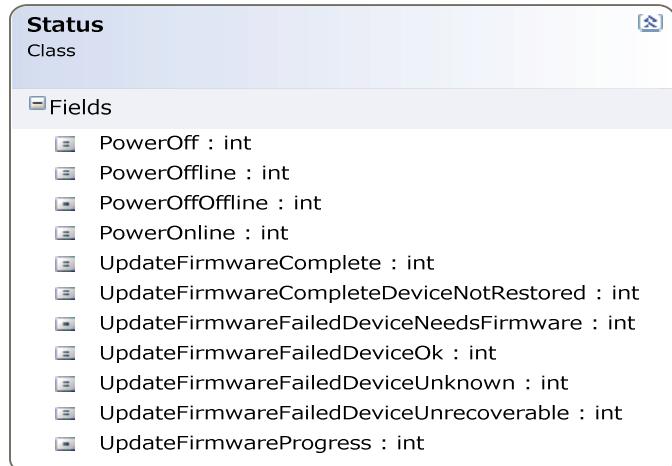
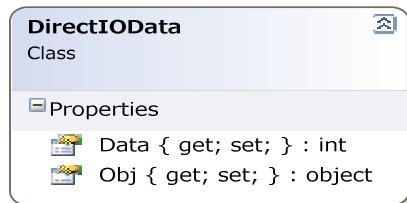


WS-POS 1.1 Technical Specification

Lights



Lights Events



Line Display

LineDisplay
Interface

Methods

- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- ClearDescriptors() : void
- ClearText() : void
- Close(string EndpointAddress) : void
 - CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
- CreateWindow(int ViewportRow, int ViewportColumn, int ViewportHeight, int ViewportWidth, int WindowHeight, int WindowWidth) : void
- DefineGlyph(int GlyphCode, byte[] Glyph) : void
- DestroyWindow() : void
- DirectIO(int Command, int Data, object Obj) : DirectIOData
 - DisplayBitmap(string FileName, int Width, int AlignmentX, int AlignmentY) : void
 - DisplayText(string Data, DisplayTextMode Attribute) : void
 - DisplayTextAt(int Row, int Column, string Data, DisplayTextMode Attribute) : void
- GetBlinkRate() : int
- GetCapBitmap() : bool
- GetCapBlink() : DisplayBlink
- GetCapBlinkRate() : bool
- GetCapBrightness() : bool
- GetCapCharacterSet() : CharacterSetCapability
- GetCapCompareFirmwareVersion() : bool
- GetCapCursorType() : int
- GetCapCustomGlyph() : bool
- GetCapDescriptors() : bool
- GetCapHMarquee() : bool
- GetCapCharWait() : bool
- GetCapMapCharacterSet() : bool
- GetCapPowerReporting() : PowerReporting
- GetCapReadBack() : DisplayReadBack
- GetCapReverse() : DisplayReverse

Line Display

The diagram shows a UML class named "LineDisplay" with the following details:

- LineDisplay** is a class under the **Interface** package.
- Methods**:
 - GetCapStatisticsReporting() : bool
 - GetCapUpdateFirmware() : bool
 - GetCapUpdateStatistics() : bool
 - GetCapVMarquee() : bool
 - GetCharacterSet() : int
 - GetCharacterSetList() : CharacterSetList
 - GetCheckHealthText() : string
 - GetClaimed() : bool
 - GetColumns() : int
 - GetCurrentWindow() : int
 - GetCursorColumn() : int
 - GetCursorRow() : int
 - GetCursorType() : int
 - GetCursorUpdate() : bool
 - GetCustomGlyphList() : CustomGlyphList
 - GetDeviceBrightness() : int
 - GetDeviceColumns() : int
 - GetDeviceControlDescription() : string
 - GetDeviceControlVersion() : UposVersion
 - GetDeviceDescriptors() : int
 - GetDeviceEnabled() : bool
 - GetDeviceRows() : int
 - GetDeviceServiceDescription() : string
 - GetDeviceServiceVersion() : UposVersion
 - GetDeviceWindows() : int
 - GetFreezeEvents() : bool
 - GetGlyphHeight() : int
 - GetGlyphWidth() : int
 - GetInterCharacterWait() : int

Line Display

The diagram shows a UML class named "LineDisplay Interface". It has a single association named "Methods" which points to a list of method signatures. The methods listed are:

- GetMapCharacterSet() : bool
- GetMarqueeFormat() : DisplayMarqueeFormat
- GetMarqueeRepeatWait() : int
- GetMarqueeType() : DisplayMarqueeType
- GetMarqueeUnitWait() : int
- GetMaximumX() : int
- GetMaximumY() : int
- GetPhysicalDeviceDescription() : string
- GetPhysicalDeviceName() : string
- GetPowerNotify() : PowerNotification
- GetPowerState() : PowerState
- GetRows() : int
- GetScreenMode() : int
- GetScreenModeList() : DisplayScreenModeList
- GetState() : ControlState
- Open(string EndpointAddress) : void
- ReadCharacterAtCursor() : int
- RefreshWindow(int Window) : void
- Release() : void
- ResetStatistics(StatisticList StatisticsBuffer) : void
- RetrieveStatistics(StatisticList StatisticsBuffer) : string
- ScrollText(DisplayScrollText Direction, int Units) : void
- SetBitmap(int BitmapNumber, string FileName, int Width, int AlignmentX, int AlignmentY) : void
- SetBlinkRate(int BlinkRate) : void
- SetCharacterSet(int CharacterSet) : void

Line Display

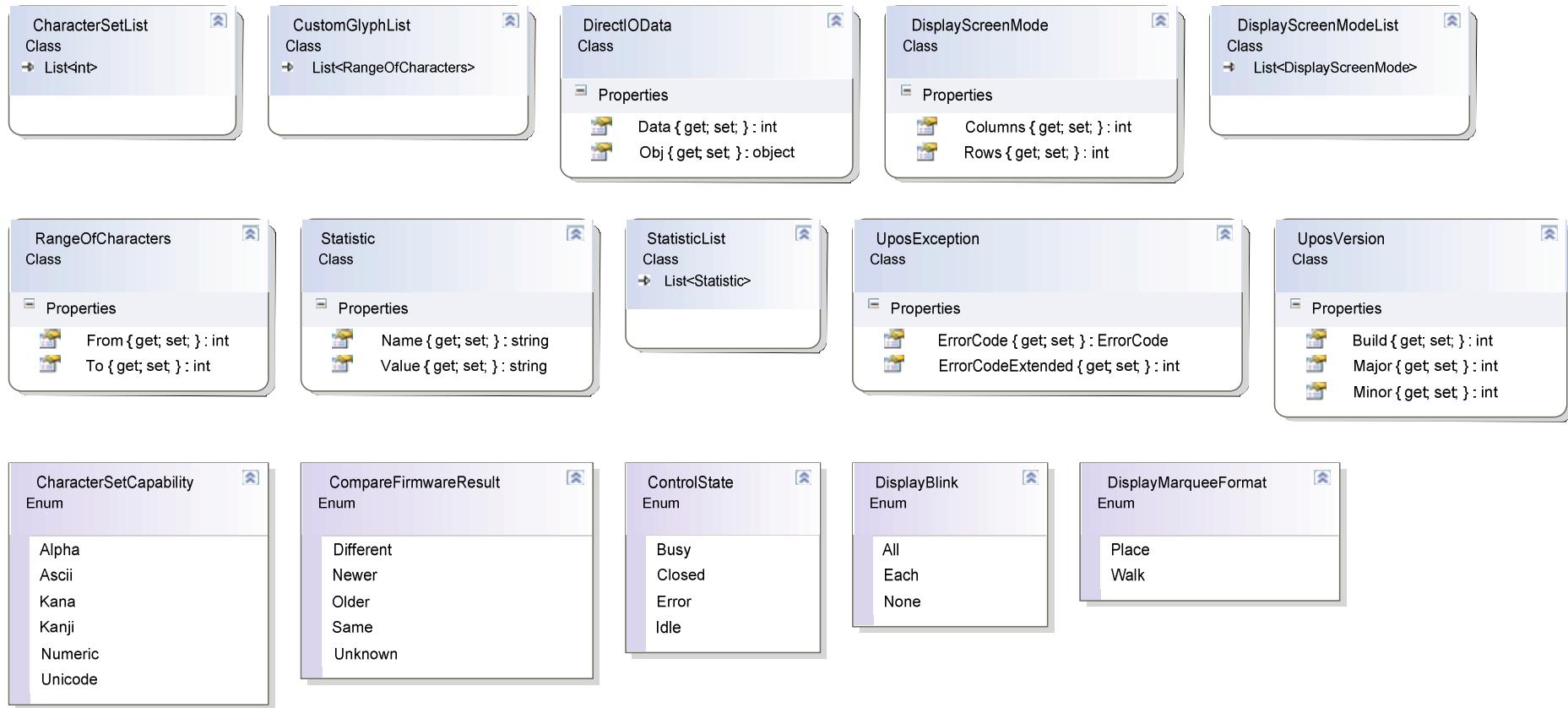


Methods

- SetCurrentWindow(int CurrentWindow) : void
- SetCursorColumn(int CursorColumn) : void
- SetCursorPosition(int CursorRow) : void
- SetCursorType(int CursorType) : void
- SetCursorUpdate(bool CursorUpdate) : void
- SetDescriptor(int Descriptor, DisplaySetDescriptor Attribute) : void
- SetDeviceBrightness(int DeviceBrightness) : void
- SetDeviceEnabled(bool DeviceEnabled) : void
- SetFreezeEvents(bool FreezeEvents) : void
- SetInterCharacterWait(int InterCharacterWait) : void
- SetMapCharacterSet(bool MapCharacterSet) : void
- SetMarqueeFormat(DisplayMarqueeFormat MarqueeFormat) : void
- SetMarqueeRepeatWait(int MarqueeRepeatWait) : void
- SetMarqueeType(DisplayMarqueeType MarqueeType) : void
- SetMarqueeUnitWait(int MarqueeUnitWait) : void
- SetPowerNotify(PowerNotification PowerNotify) : void
- SetScreenMode(int ScreenMode) : void
- UpdateFirmware(string FirmwareFileName) : void
- UpdateStatistics(StatisticList StatisticsBuffer) : void

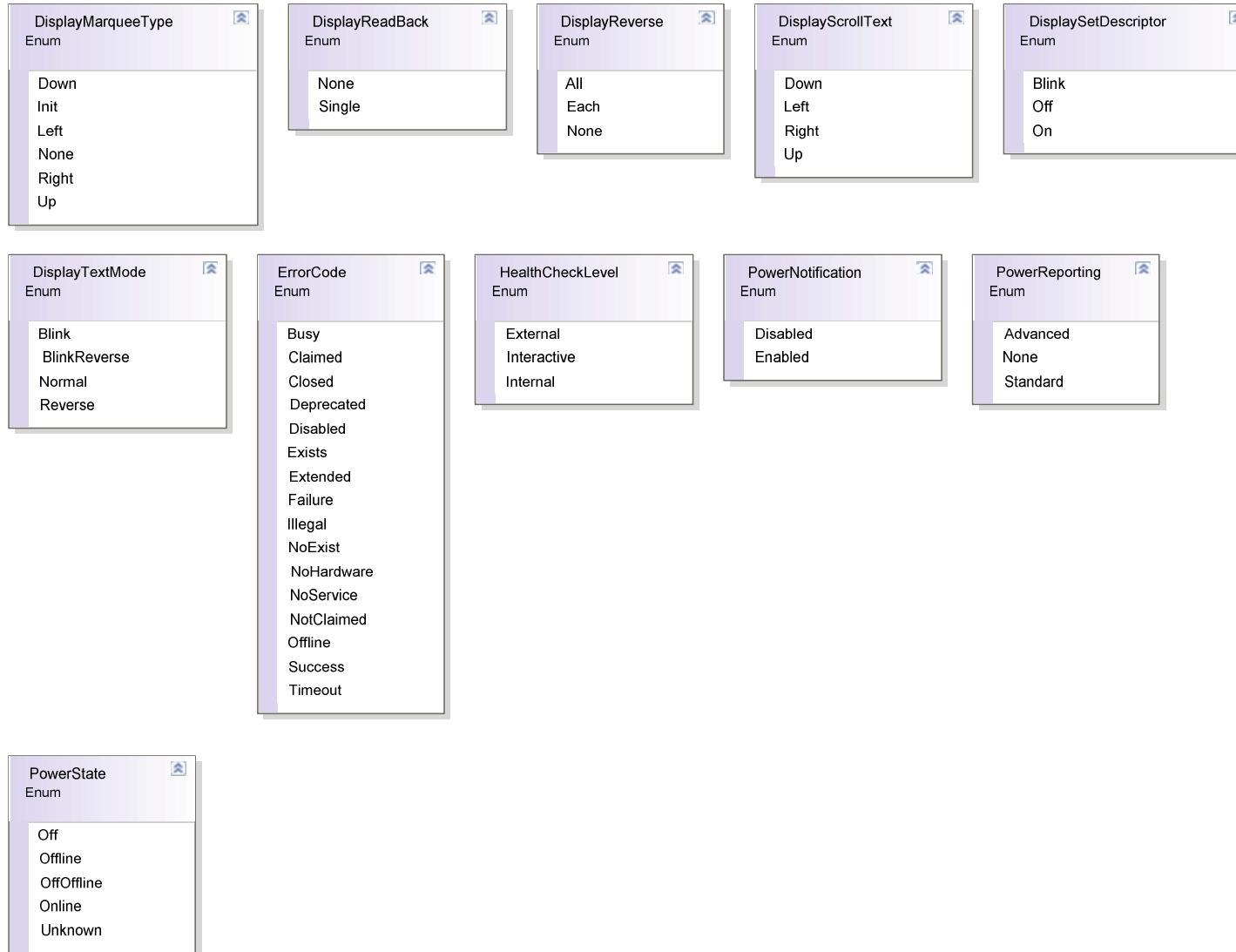
WS-POS 1.1 Technical Specification

Line Display



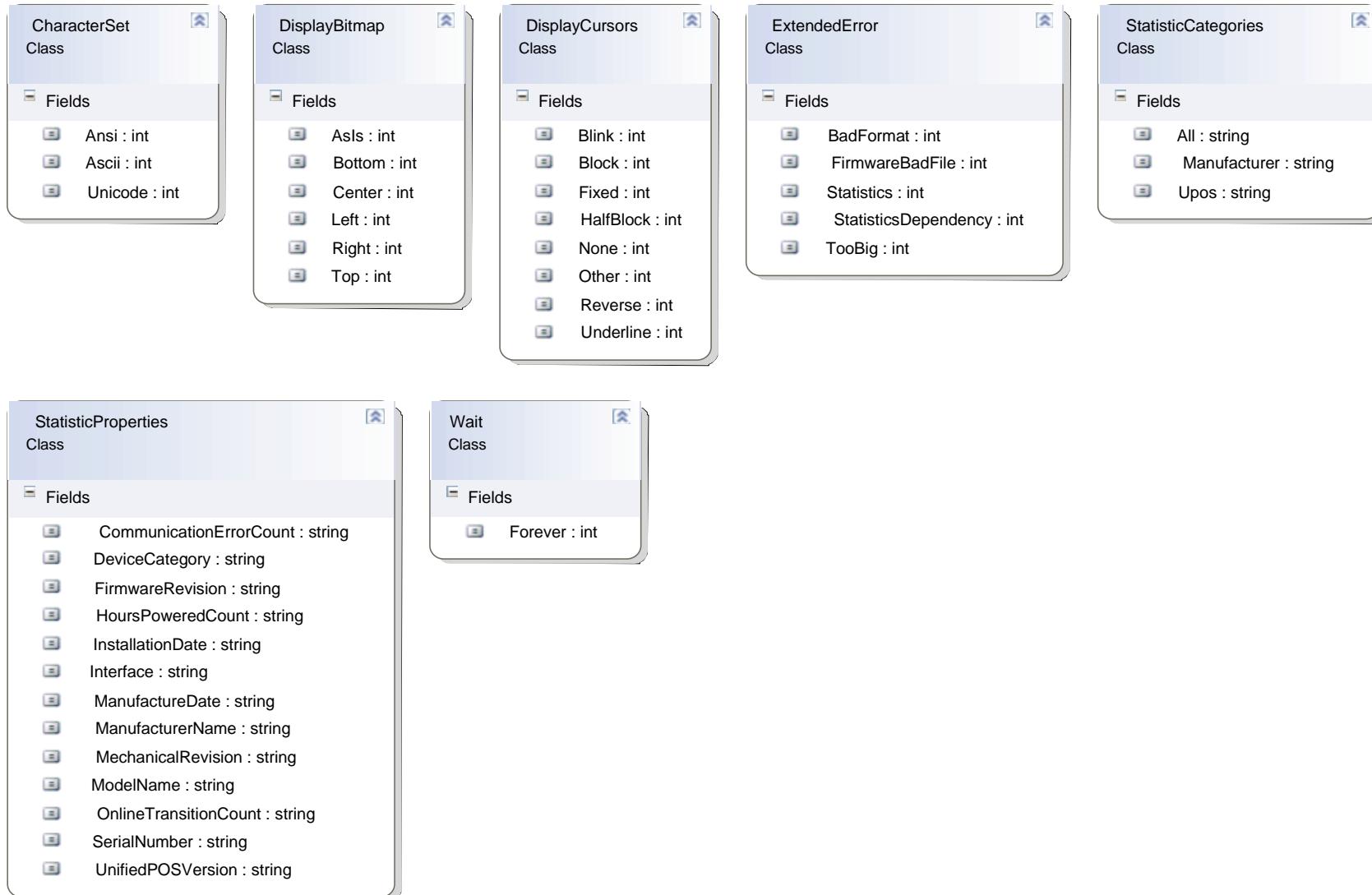
WS-POS 1.1 Technical Specification

Line Display

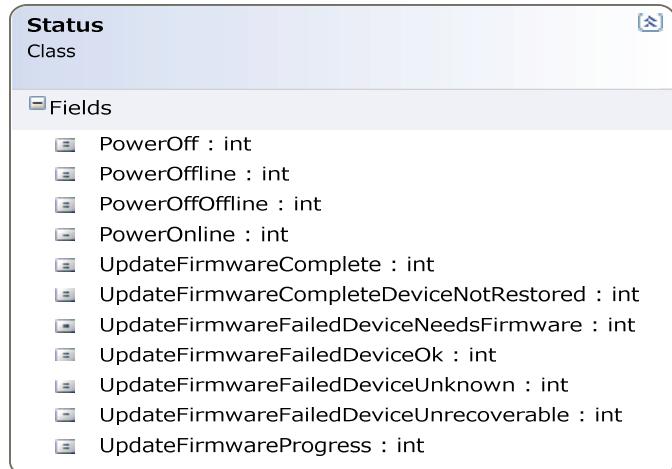
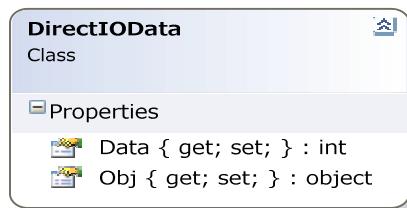


WS-POS 1.1 Technical Specification

Line Display



Line Display Events



Magnetic Ink Character Recognition (MICR)

MICR Interface

Methods

- BeginInsertion(int Timeout) : void
- BeginRemoval(int Timeout) : void
- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- ClearInput() : void
- ClearInputProperties() : void
- Close(string EndpointAddress) : void
 - CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
- DirectIO(int Command, int Data, object Obj) : DirectIOData
- EndInsertion() : void
- EndRemoval() : void
- GetAccountNumber() : string
- GetAmount() : string
- GetAutoDisable() : bool
- GetBankNumber() : string
- GetCapCompareFirmwareVersion() : bool
- GetCapPowerReporting() : PowerReporting
- GetCapStatisticsReporting() : bool
- GetCapUpdateFirmware() : bool
- GetCapUpdateStatistics() : bool
- GetCapValidationDevice() : bool

Magnetic Ink Character Recognition (MICR)



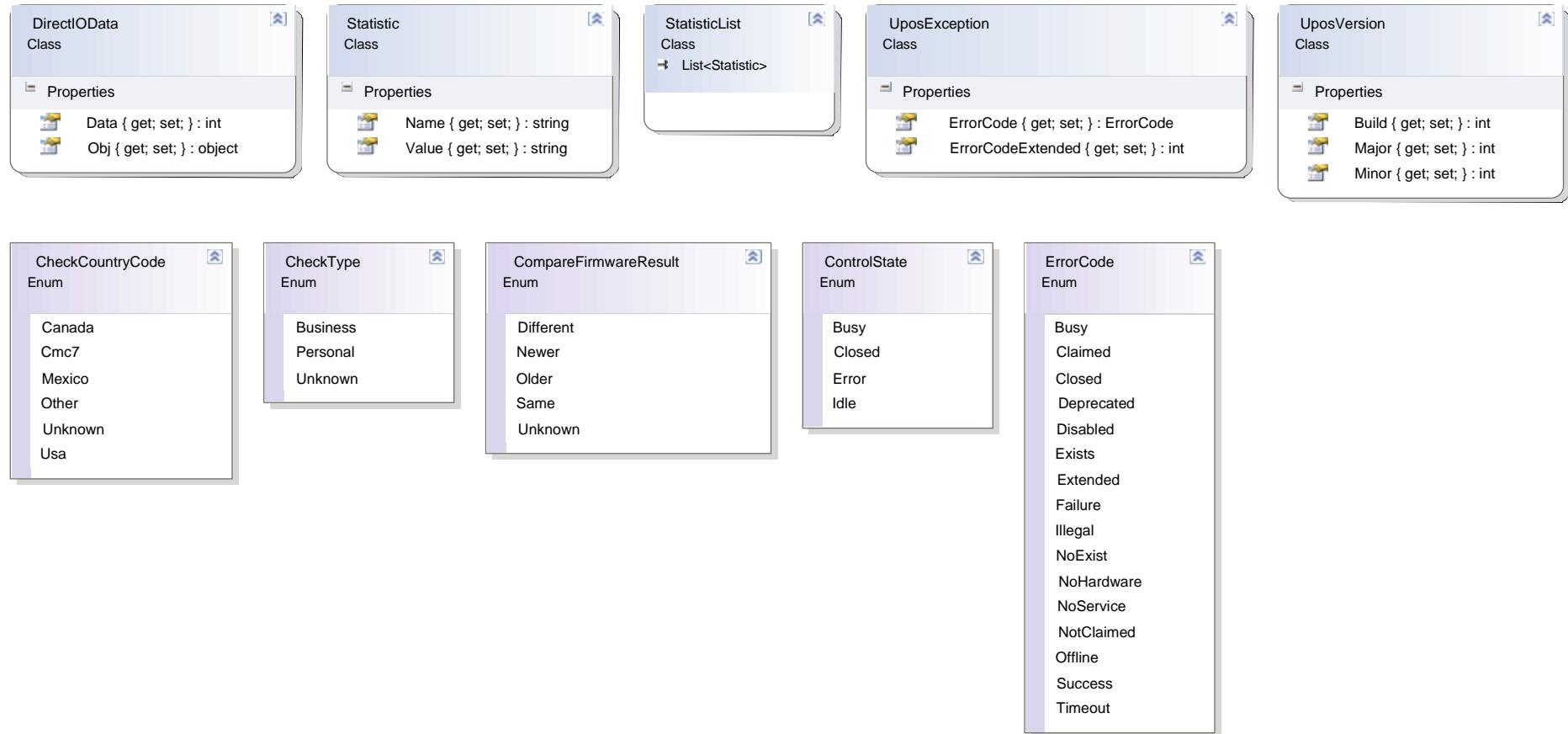
- ⌚ GetCheckHealthText() : string
- ⌚ GetCheckType() : CheckType
- ⌚ GetClaimed() : bool
- ⌚ GetCountryCode() : CheckCountryCode
- ⌚ GetDataCount() : int
- ⌚ GetDataEventEnabled() : bool
- ⌚ GetDeviceControlDescription() : string
- ⌚ GetDeviceControlVersion() : UposVersion
- ⌚ GetDeviceEnabled() : bool
- ⌚ GetDeviceServiceDescription() : string
- ⌚ GetDeviceServiceVersion() : UposVersion
- ⌚ GetEPC() : string
- ⌚ GetFreezeEvents() : bool
- ⌚ GetPhysicalDeviceDescription() : string
- ⌚ GetPhysicalDeviceName() : string
- ⌚ GetPowerNotify() : PowerNotification
- ⌚ GetPowerState() : PowerState
- ⌚ GetRawData() : string

Magnetic Ink Character Recognition (MICR)



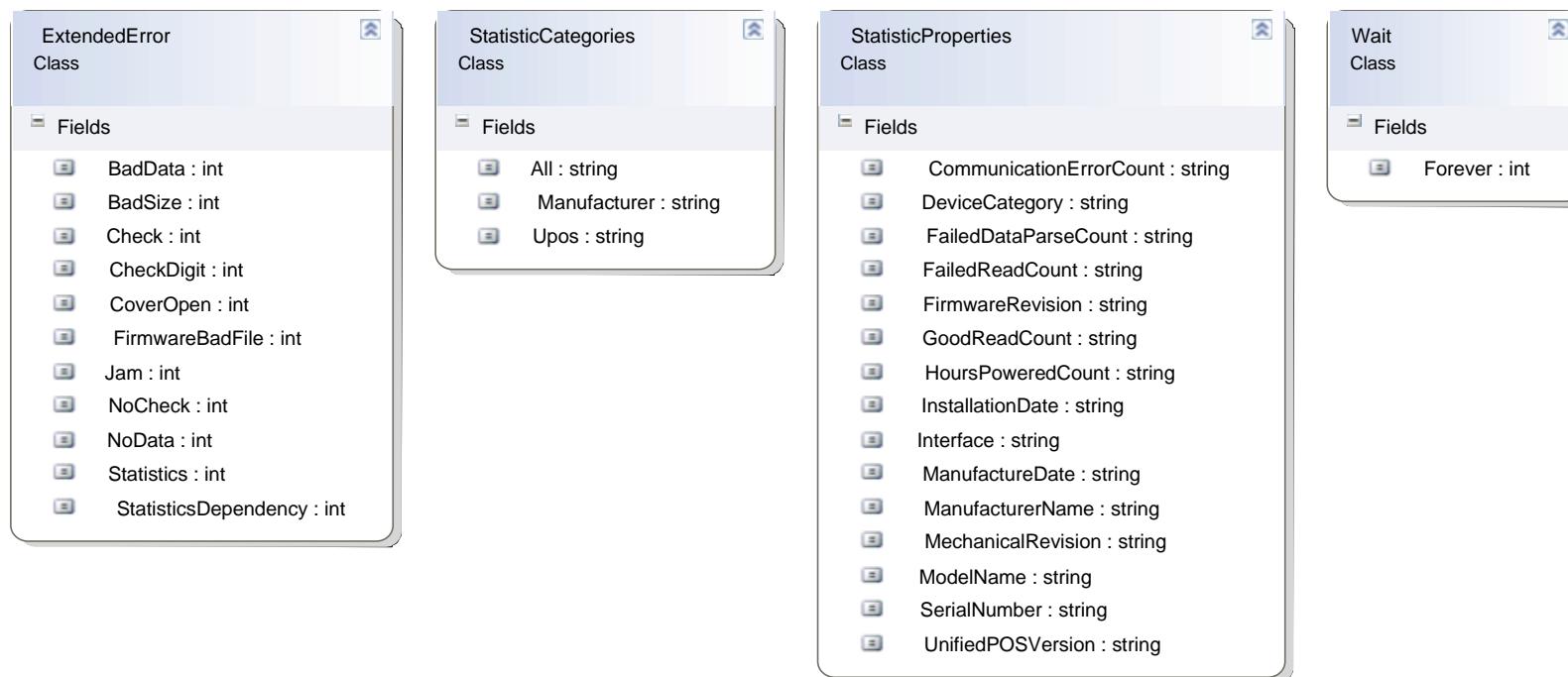
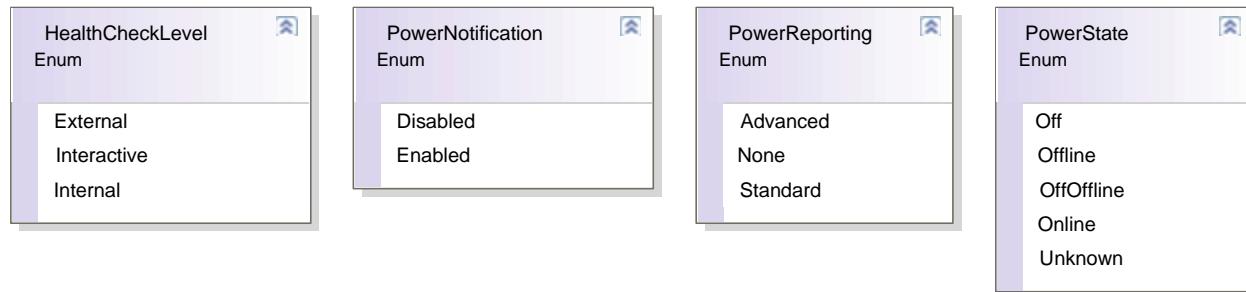
- GetSerialNumber() : string
- GetState() : ControlState
- GetTransitNumber() : string
- Open(string EndpointAddress) : void
- Release() : void
- ResetStatistics(StatisticList StatisticsBuffer) : void
- RetrieveStatistics(StatisticList StatisticsBuffer) : string
- SetAutoDisable(bool AutoDisable) : void
- SetDataEventEnabled(bool DataEventEnabled) : void
- SetDeviceEnabled(bool DeviceEnabled) : void
- SetFreezeEvents(bool FreezeEvents) : void
- SetPowerNotify(PowerNotification PowerNotify) : void
- UpdateFirmware(string FirmwareFileName) : void
- UpdateStatistics(StatisticList StatisticsBuffer) : void

Magnetic Ink Character Recognition (MICR)



WS-POS 1.1 Technical Specification

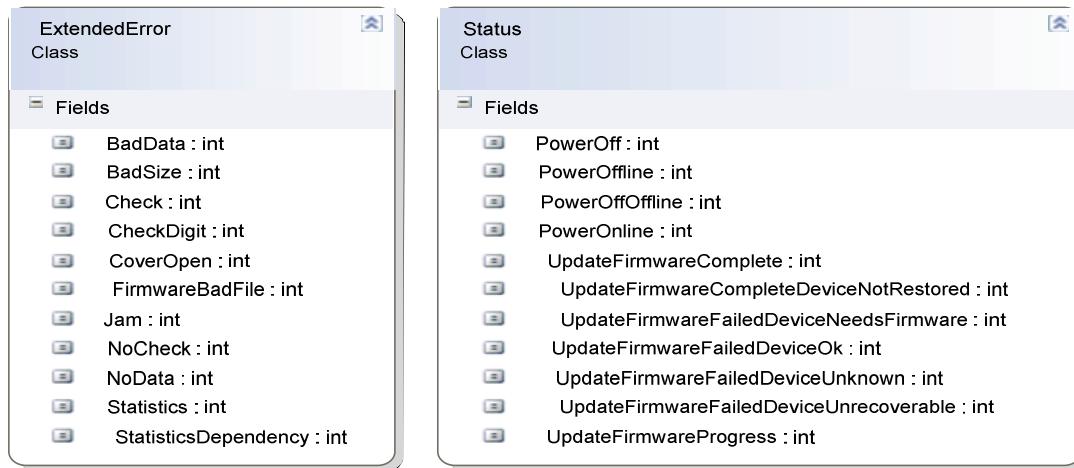
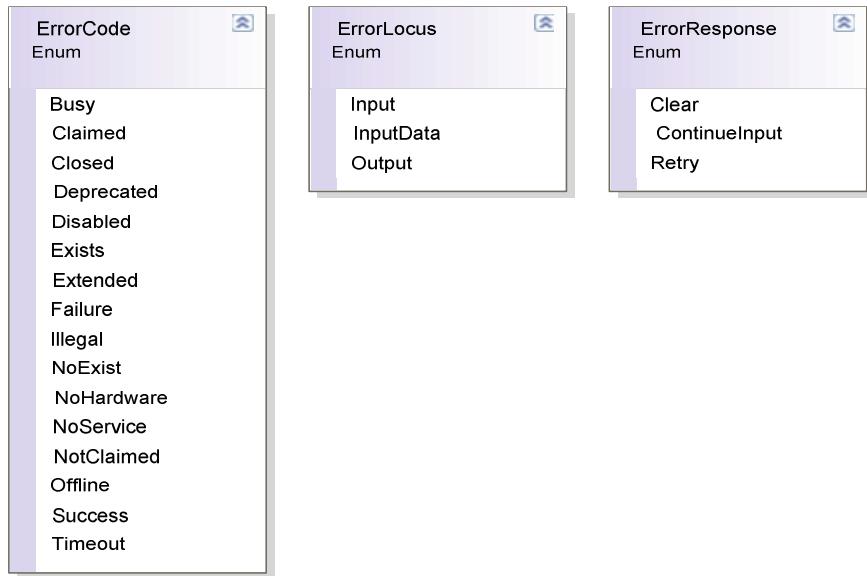
Magnetic Ink Character Recognition (MICR)



Magnetic Ink Character Recognition (MICR) Events



Magnetic Ink Character Recognition (MICR) Events



Motion Sensor

The diagram shows a UML class named "MotionSensor Interface". It has a single section labeled "Methods" which contains the following list of methods:

- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- Close(string EndpointAddress) : void
- CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
- DirectIO(int Command, int Data, object Obj) : DirectIOData
- GetCapCompareFirmwareVersion() : bool
- GetCapPowerReporting() : PowerReporting
- GetCapStatisticsReporting() : bool
- GetCapUpdateFirmware() : bool
- GetCapUpdateStatistics() : bool
- GetCheckHealthText() : string
- GetClaimed() : bool
- GetDeviceControlDescription() : string
- GetDeviceControlVersion() : UposVersion
- GetDeviceEnabled() : bool
- GetDeviceServiceDescription() : string
- GetDeviceServiceVersion() : UposVersion
- GetFreezeEvents() : bool
- GetMotion() : bool

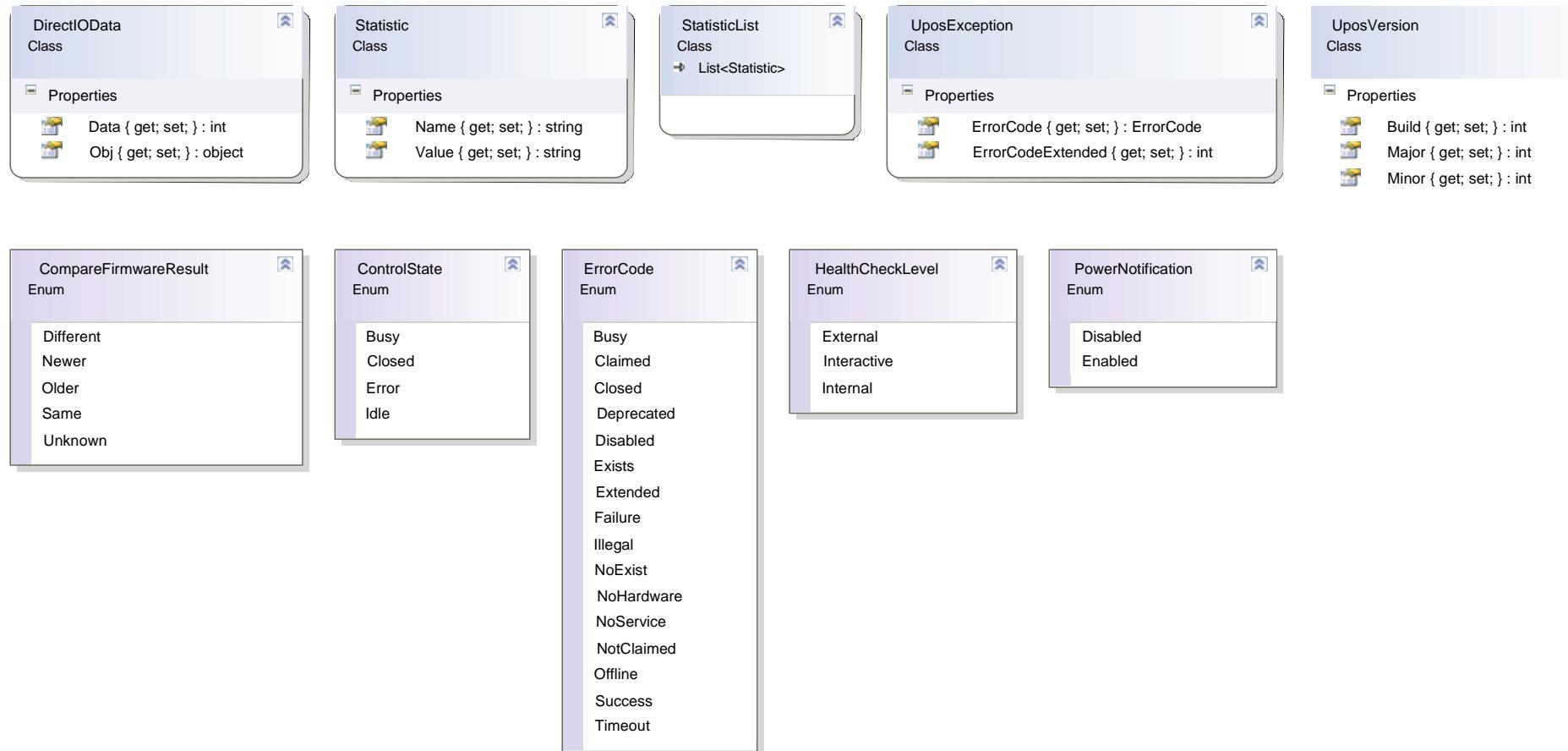
Motion Sensor



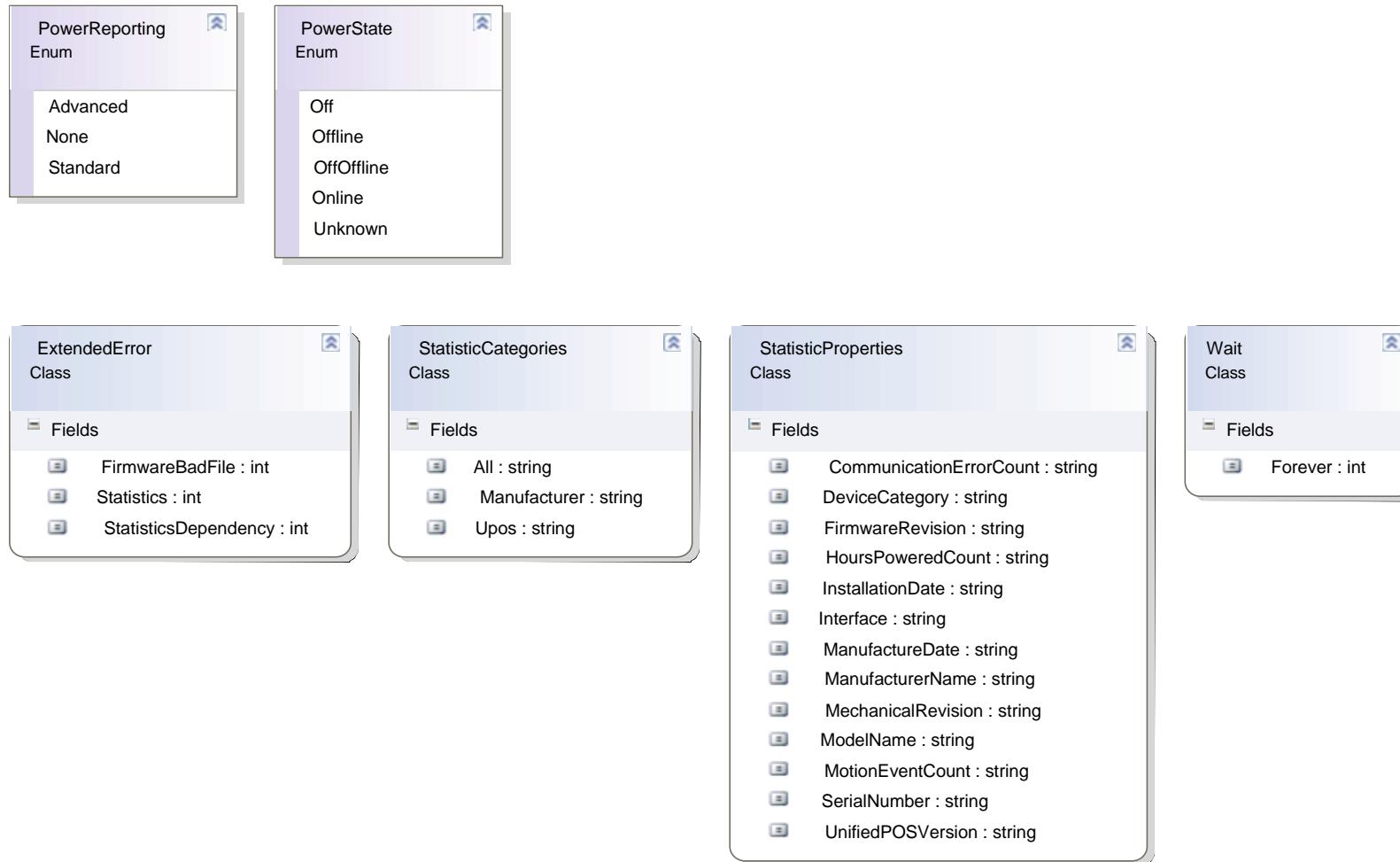
Methods

- GetPhysicalDeviceDescription() : string
- GetPhysicalDeviceName() : string
- GetPowerNotify() : PowerNotification
- GetPowerState() : PowerState
- GetState() : ControlState
- GetTimeout() : int
- Open(string EndpointAddress) : void
- Release() : void
- ResetStatistics(StatisticList StatisticsBuffer) : void
- RetrieveStatistics(StatisticList StatisticsBuffer) : string
- SetDeviceEnabled(bool DeviceEnabled) : void
- SetFreezeEvents(bool FreezeEvents) : void
- SetPowerNotify(PowerNotification PowerNotify) : void
- SetTimeout(int Timeout) : void
- UpdateFirmware(string FirmwareFileName) : void
- UpdateStatistics(StatisticList StatisticsBuffer) : void
- WaitForMotion(int Timeout) : void

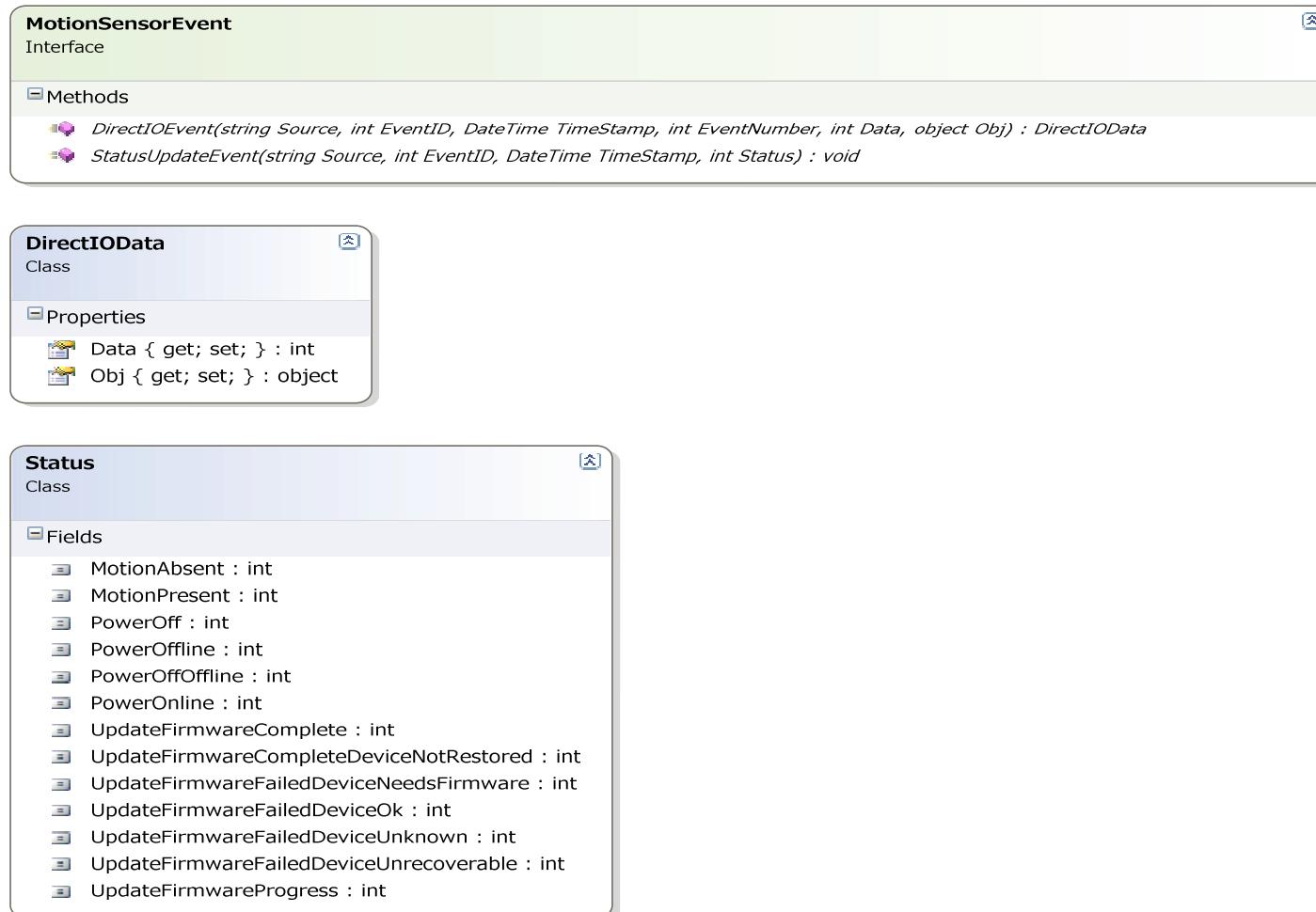
Motion Sensor



Motion Sensor



Motion Sensor Events



Magnetic Stripe Reader (MSR)

The diagram shows a UML class named 'MSR Interface' with a light green background. It has a section titled 'Methods' containing a list of 34 methods, each preceded by a small purple icon. The methods are:

- AuthenticateDevice(byte[] Response) : void
- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- ClearInput() : void
- ClearInputProperties() : void
- Close(string EndpointAddress) : void
- CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
- DeauthenticateDevice(byte[] Response) : void
- DirectIO(int Command, int Data, object Obj) : DirectIOData
- GetAccountNumber() : string
- GetAdditionalSecurityInformation() : byte[]
- GetAutoDisable() : bool
- GetCapCardAuthentication() : string
- GetCapCompareFirmwareVersion() : bool
- GetCapDataEncryption() : int
- GetCapDeviceAuthentication() : DeviceAuthenticationLevel
- GetCapISO() : bool
- GetCapJISOne() : bool
- GetCapJISTwo() : bool
- GetCapPowerReporting() : PowerReporting
- GetCapStatisticsReporting() : bool
- GetCapTrackDataMasking() : bool
- GetCapTransmitSentinels() : bool
- GetCapUpdateFirmware() : bool
- GetCapUpdateStatistics() : bool
- GetCapWritableTracks() : MSRTracks
- GetCardAuthenticationData() : byte[]
- GetCardAuthenticationDataLength() : int
- GetCardPropertyList() : CardPropertyList
- GetCardType() : string
- GetCardTypeList() : CardTypeList

Magnetic Stripe Reader (MSR)



Methods

- GetCheckHealthText() : string
- GetClaimed() : bool
- GetDataCount() : int
- GetDataEncryptionAlgorithm() : int
- GetDataEventEnabled() : bool
- GetDecodeData() : bool
- GetDeviceAuthenticated() : bool
- GetDeviceAuthenticationProtocol() : DeviceAuthenticationProtocol
- GetDeviceControlDescription() : string
- GetDeviceControlVersion() : UposVersion
- GetDeviceEnabled() : bool
- GetDeviceServiceDescription() : string
- GetDeviceServiceVersion() : UposVersion
- GetEncodingMaxLength() : int
- GetErrorReportingType() : MSRErrorReporting
- GetExpirationDate() : string
- GetFirstName() : string
- GetFreezeEvents() : bool
- GetMiddleInitial() : string
- GetParseDecodeData() : bool
- GetPhysicalDeviceDescription() : string
- GetPhysicalDeviceName() : string
- GetPowerNotify() : PowerNotification
- GetPowerState() : PowerState
- GetServiceCode() : string

Magnetic Stripe Reader (MSR)



Methods

- ⊕ GetState() : ControlState
- ⊕ GetSuffix() : string
- ⊕ GetSurname() : string
- ⊕ GetTitle() : string
- ⊕ GetTrack1Data() : byte[]
- ⊕ GetTrack1DiscretionaryData() : byte[]
- ⊕ GetTrack1EncryptedData() : byte[]
- ⊕ GetTrack1EncryptedDataLength() : int
- ⊕ GetTrack2Data() : byte[]
- ⊕ GetTrack2DiscretionaryData() : byte[]
- ⊕ GetTrack2EncryptedData() : byte[]
- ⊕ GetTrack2EncryptedDataLength() : int
- ⊕ GetTrack3Data() : byte[]
- ⊕ GetTrack3EncryptedData() : byte[]
- ⊕ GetTrack3EncryptedDataLength() : int
- ⊕ GetTrack4Data() : byte[]
- ⊕ GetTrack4EncryptedData() : byte[]
- ⊕ GetTrack4EncryptedDataLength() : int
- ⊕ GetTracksToRead() : MSRTracks
- ⊕ GetTracksToWrite() : MSRTracks
- ⊕ GetTransmitSentinels() : bool
- ⊕ GetWriteCardType() : string
- ⊕ Open(string EndpointAddress) : void
- ⊕ Release() : void

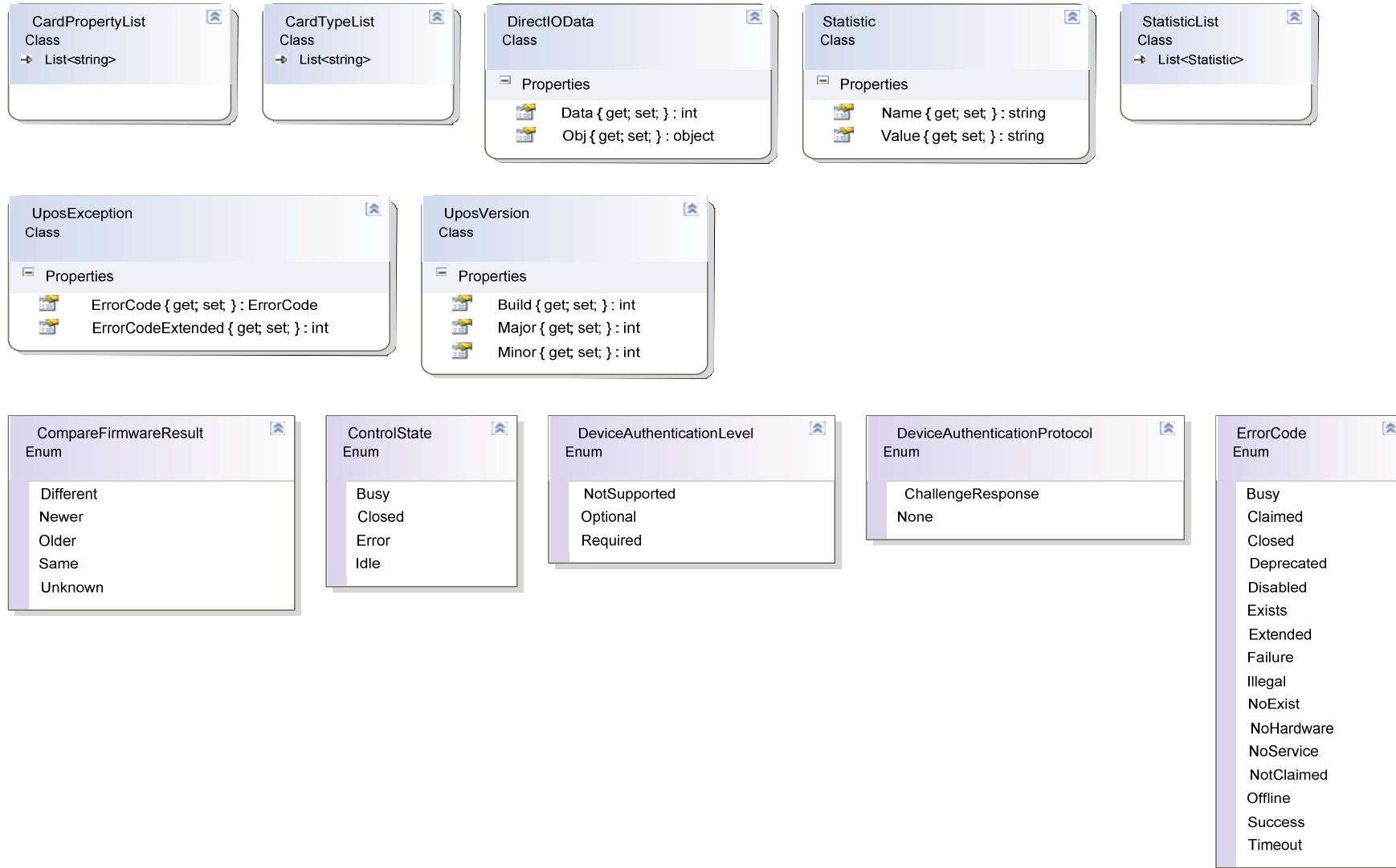
Magnetic Stripe Reader (MSR)



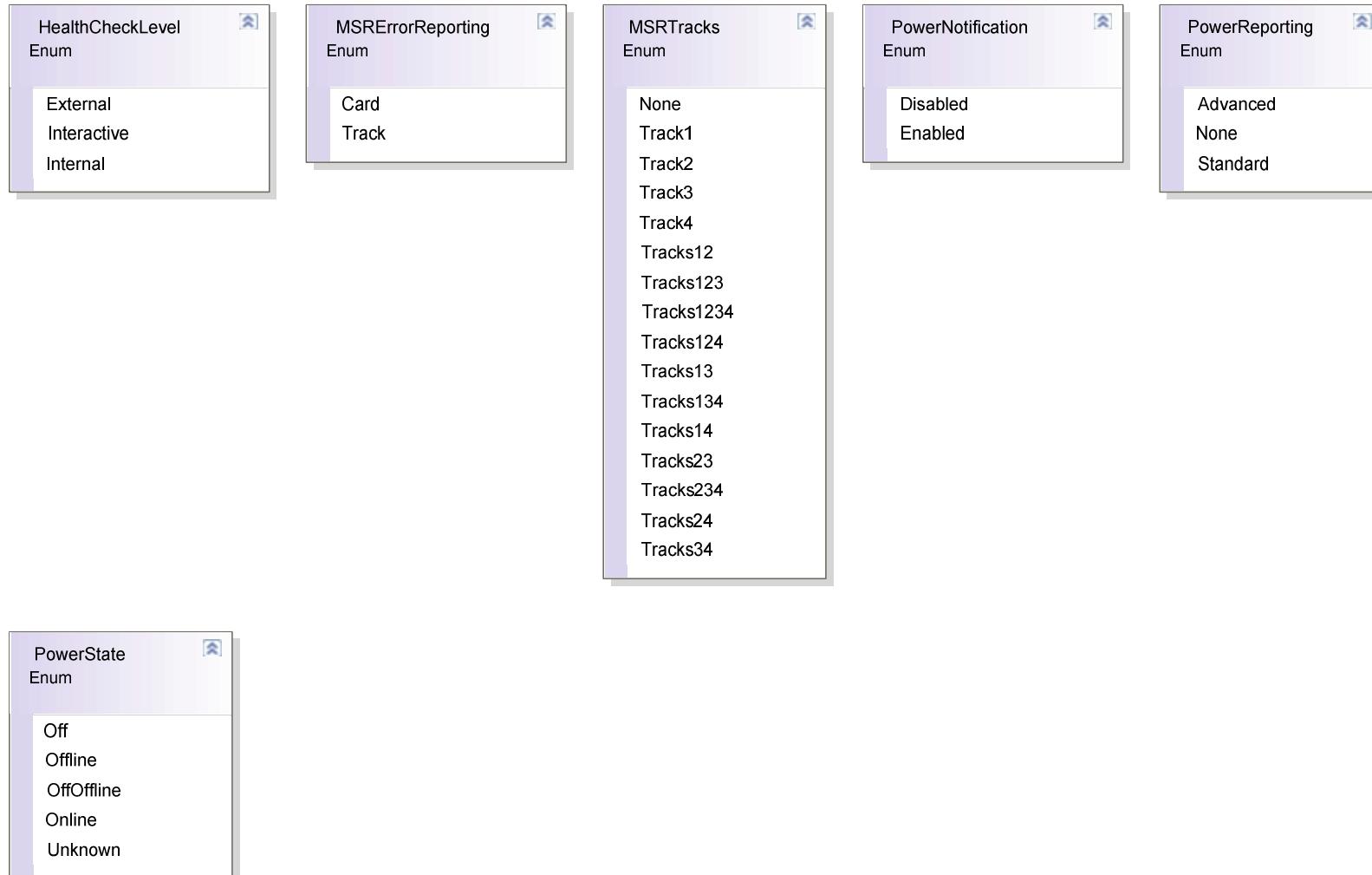
Methods

- ResetStatistics(StatisticList StatisticsBuffer) : void
- RetrieveCardProperty(string Name) : string
- RetrieveDeviceAuthenticationData() : byte[]
- RetrieveStatistics(StatisticList StatisticsBuffer) : string
- SetAutoDisable(bool AutoDisable) : void
- SetDataEncryptionAlgorithm(int DataEncryptionAlgorithm) : void
- SetDataEventEnabled(bool DataEventEnabled) : void
- SetDecodeData(bool DecodeData) : void
- SetDeviceEnabled(bool DeviceEnabled) : void
 - SetErrorReportingType(MSRErrorReporting ErrorReportingType) : void
- SetFreezeEvents(bool FreezeEvents) : void
- SetParseDecodeData(bool ParseDecodeData) : void
- SetPowerNotify(PowerNotification PowerNotify) : void
- SetTracksToRead(MSRTracks TracksToRead) : void
- SetTracksToWrite(MSRTracks TracksToWrite) : void
- SetTransmitSentinels(bool TransmitSentinels) : void
- SetWriteCardType(string WriteCardType) : void
- UpdateFirmware(string FirmwareFileName) : void
- UpdateKey(string Key, string KeyName) : void
- UpdateStatistics(StatisticList StatisticsBuffer) : void
- WriteTracks(byte[] Track1Data, byte[] Track2Data, byte[] Track3Data, byte[] Track4Data, int Timeout) : void

Magnetic Stripe Reader (MSR)



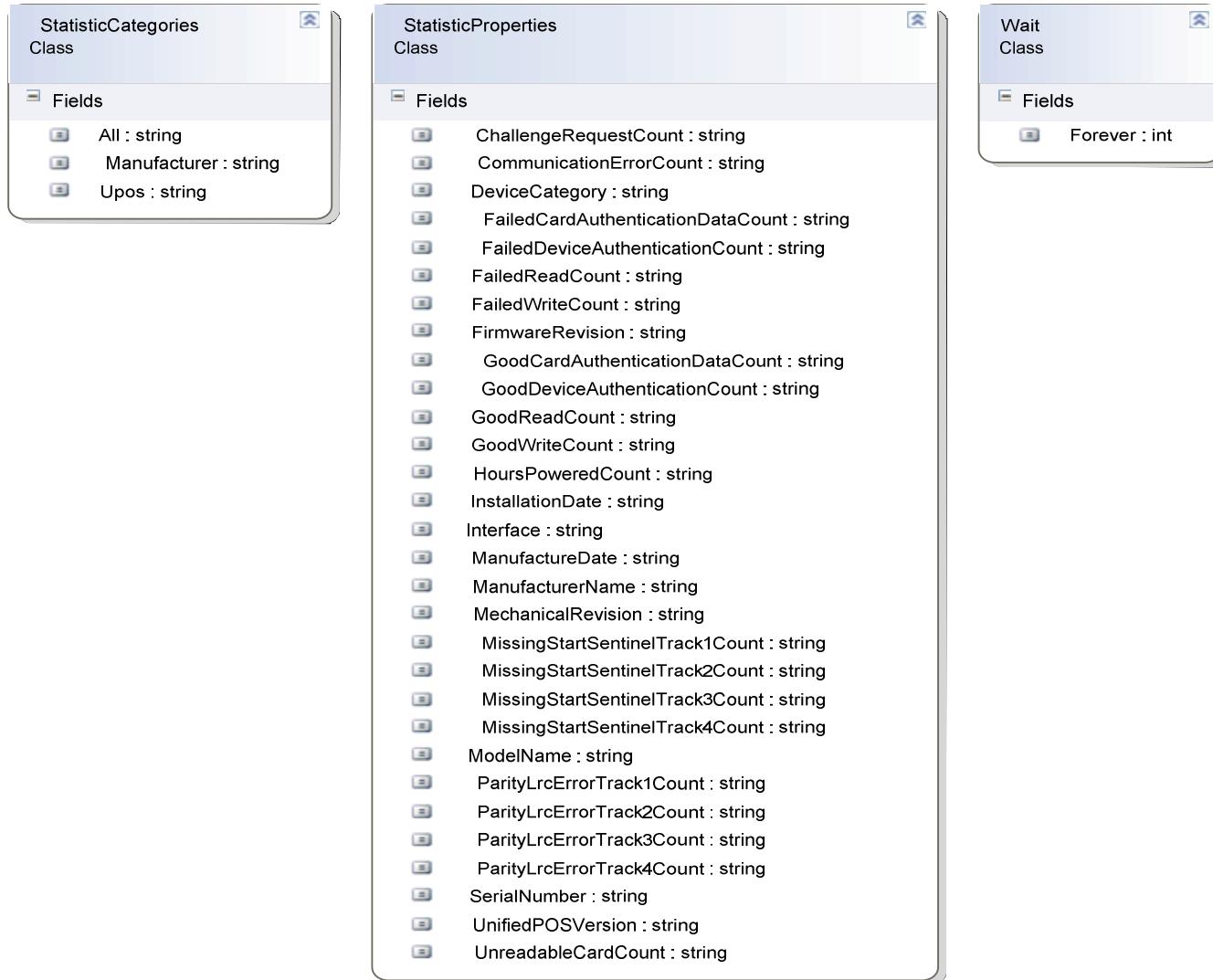
Magnetic Stripe Reader (MSR)



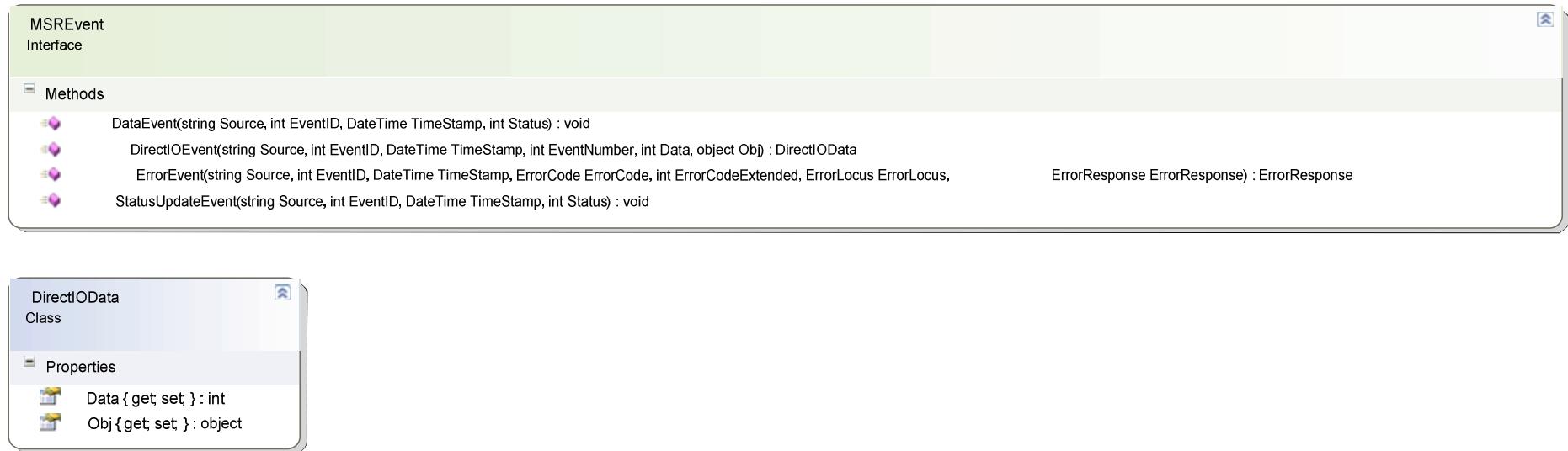
Magnetic Stripe Reader (MSR)



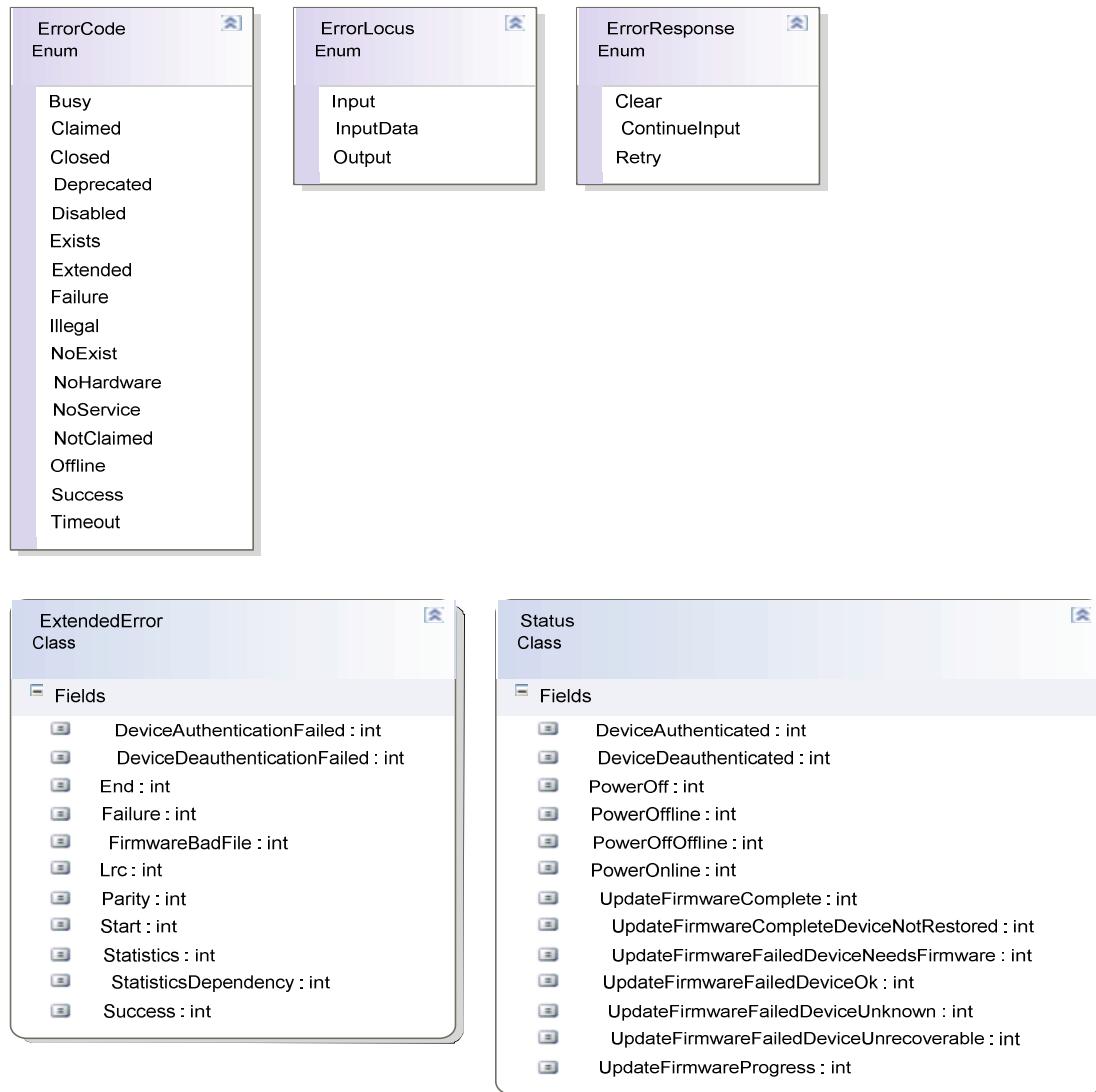
Magnetic Stripe Reader (MSR)



Magnetic Stripe Reader (MSR) Events



Magnetic Stripe Reader (MSR) Events



Personal Identification Number Pad (PIN Pad)

PINPad
Interface

Methods

- BeginEFTTransaction(PINPadSystem PINPadSystem, int TransactionHost) : void
- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- ClearInput() : void
- ClearInputProperties() : void
- Close(string EndpointAddress) : void
 - CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
- ComputeMAC(string InMsg) : string
- DirectIO(int Command, int Data, object Obj) : DirectIOData
- EnablePINEntry() : void
- EndEFTTransaction(EFTTransactionCompletion CompletionCode) : void
- GetAccountNumber() : string
- GetAdditionalSecurityInformation() : string
- GetAmount() : decimal
- GetAvailableLanguagesList() : LanguageList
- GetAvailablePromptsList() : PINPadMessageList
- GetCapCompareFirmwareVersion() : bool
- GetCapDisplay() : PINPadDisplay
- GetCapKeyboard() : bool
- GetCapLanguage() : PINPadLanguage
- GetCapMACCalculation() : bool
- GetCapPowerReporting() : PowerReporting
- GetCapStatisticsReporting() : bool
- GetCapTone() : bool
- GetCapUpdateFirmware() : bool
- GetCapUpdateStatistics() : bool
- GetCheckHealthText() : string
- GetClaimed() : bool

Personal Identification Number Pad (PIN Pad)

The diagram shows a UML class named "PINPad Interface". It has a single section labeled "Methods" which contains the following list of methods:

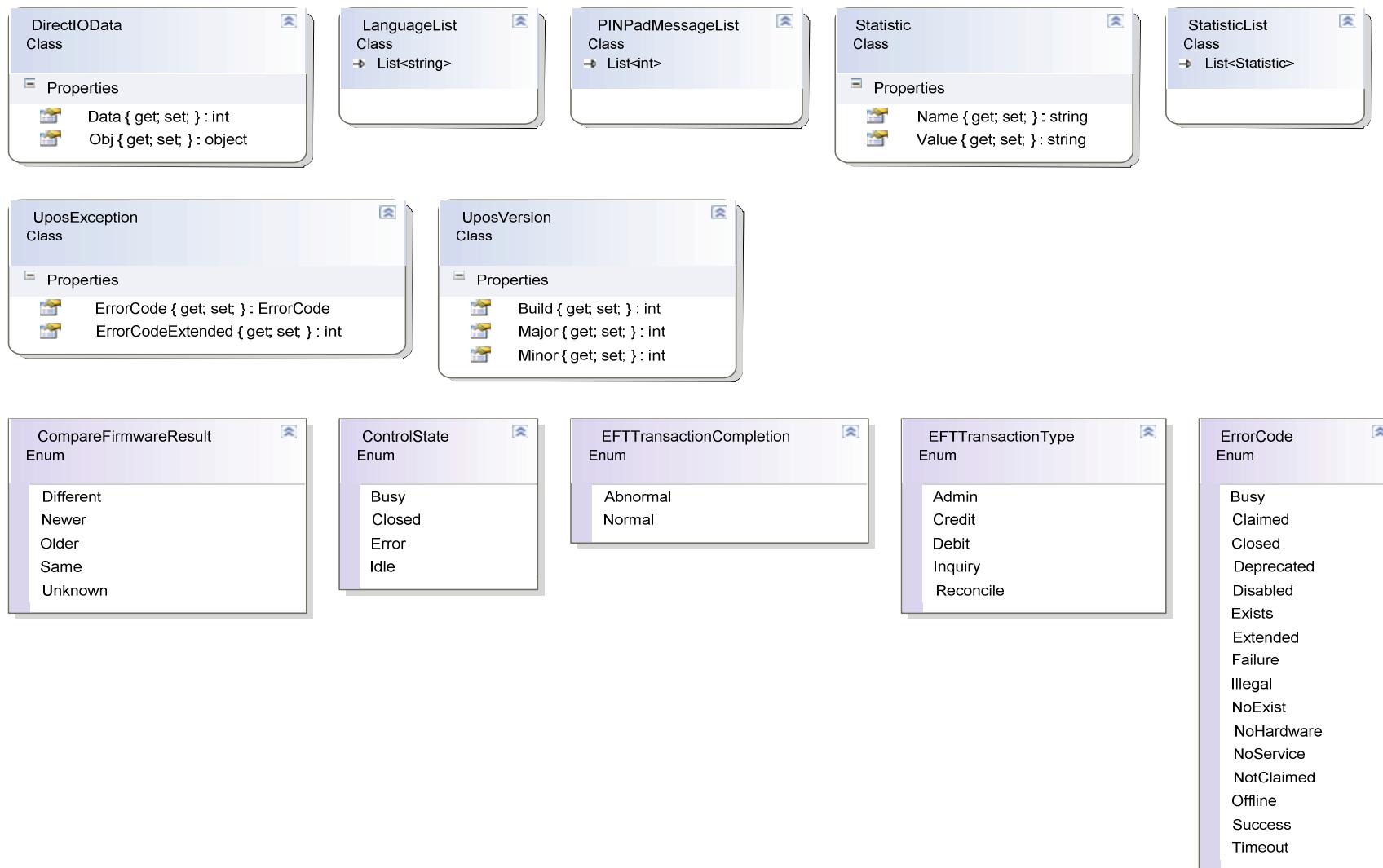
- GetDataCount() : int
- GetDataEventEnabled() : bool
- GetDeviceControlDescription() : string
- GetDeviceControlVersion() : UposVersion
- GetDeviceEnabled() : bool
- GetDeviceServiceDescription() : string
- GetDeviceServiceVersion() : UposVersion
- GetEncryptedPIN() : string
- GetFreezeEvents() : bool
- GetMaximumPINLength() : int
- GetMerchantID() : string
- GetMinimumPINLength() : int
- GetPhysicalDeviceDescription() : string
- GetPhysicalDeviceName() : string
- GetPINEEntryEnabled() : bool
- GetPowerNotify() : PowerNotification
- GetPowerState() : PowerState
- GetPrompt() : int
- GetPromptLanguage() : string
- GetState() : ControlState
- GetTerminalID() : string
- GetTrack1Data() : byte[]
- GetTrack2Data() : byte[]
- GetTrack3Data() : byte[]
- GetTrack4Data() : byte[]
- GetTransactionType() : EFTTransactionType
- Open(string EndpointAddress) : void

Personal Identification Number Pad (PIN Pad)

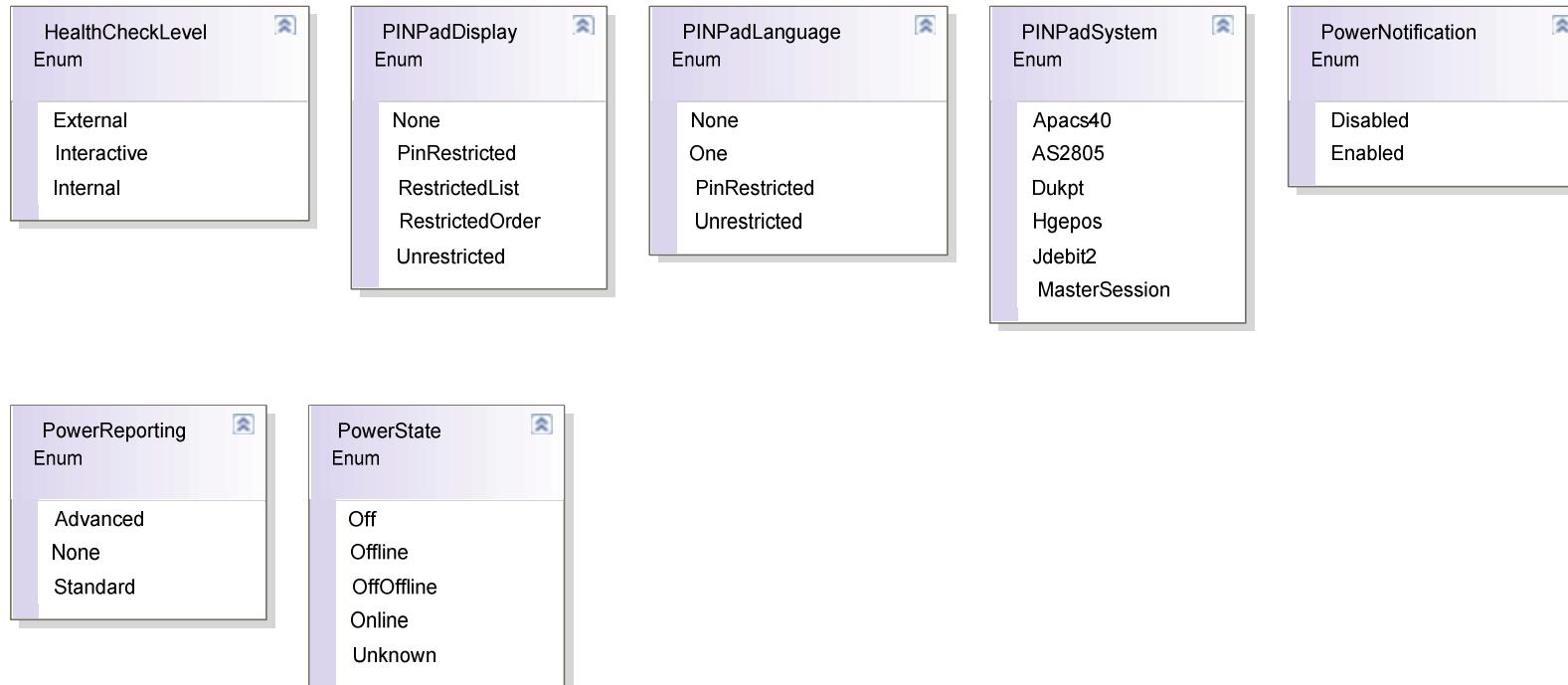


- Release() : void
- ResetStatistics(StatisticList StatisticsBuffer) : void
- RetrieveStatistics(StatisticList StatisticsBuffer) : string
- SetAccountNumber(string AccountNumber) : void
- SetAmount(decimal Amount) : void
- SetDataEventEnabled(bool DataEventEnabled) : void
- SetDeviceEnabled(bool DeviceEnabled) : void
- SetFreezeEvents(bool FreezeEvents) : void
- SetMaximumPINLength(int MaximumPINLength) : void
- SetMerchantID(string MerchantID) : void
- SetMinimumPINLength(int MinimumPINLength) : void
- SetPowerNotify(PowerNotification PowerNotify) : void
- SetPrompt(int Prompt) : void
- SetPromptLanguage(string PromptLanguage) : void
- SetTerminalID(string TerminalID) : void
- SetTrack1Data(byte[] Track1Data) : void
- SetTrack2Data(byte[] Track2Data) : void
- SetTrack3Data(byte[] Track3Data) : void
- SetTrack4Data(byte[] Track4Data) : void
- SetTransactionType(EFTTransactionType TransactionType) : void
- UpdateFirmware(string FirmwareFileName) : void
- UpdateKey(int KeyNum, string Key) : void
- UpdateStatistics(StatisticList StatisticsBuffer) : void
- VerifyMAC(string Message) : void

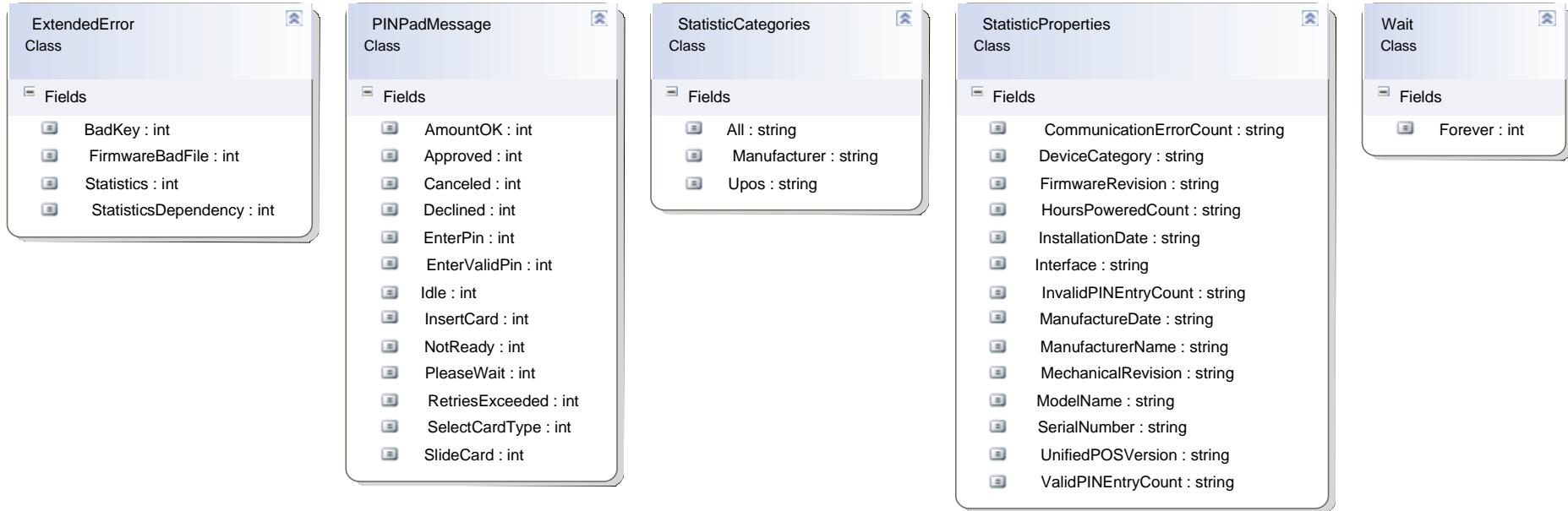
Personal Identification Number Pad (PIN Pad)



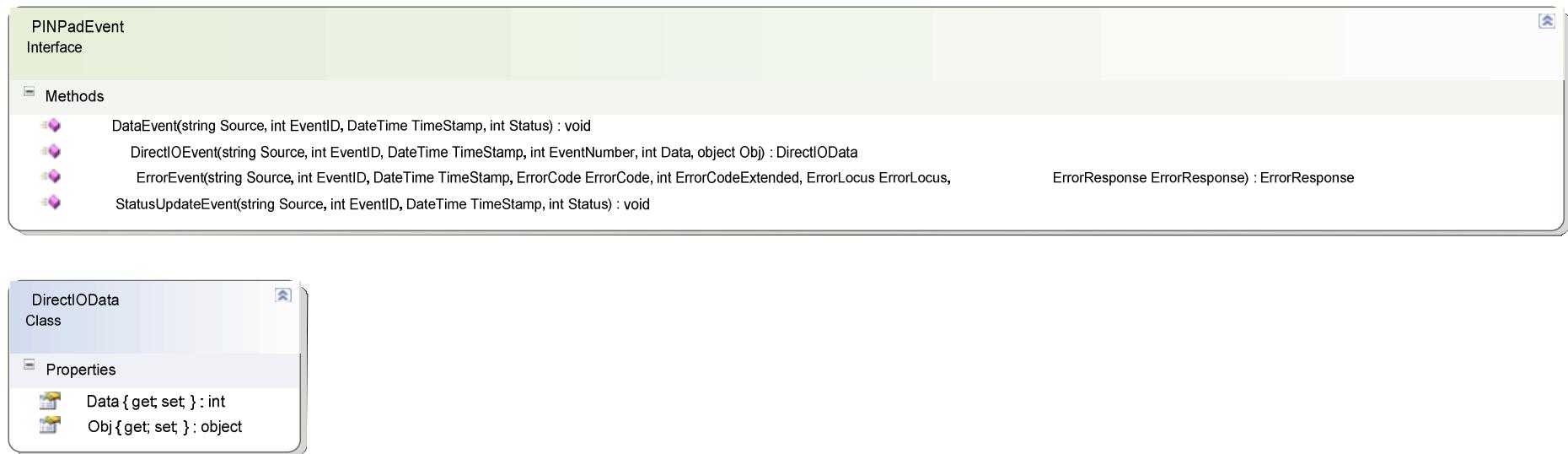
Personal Identification Number Pad (PIN Pad)



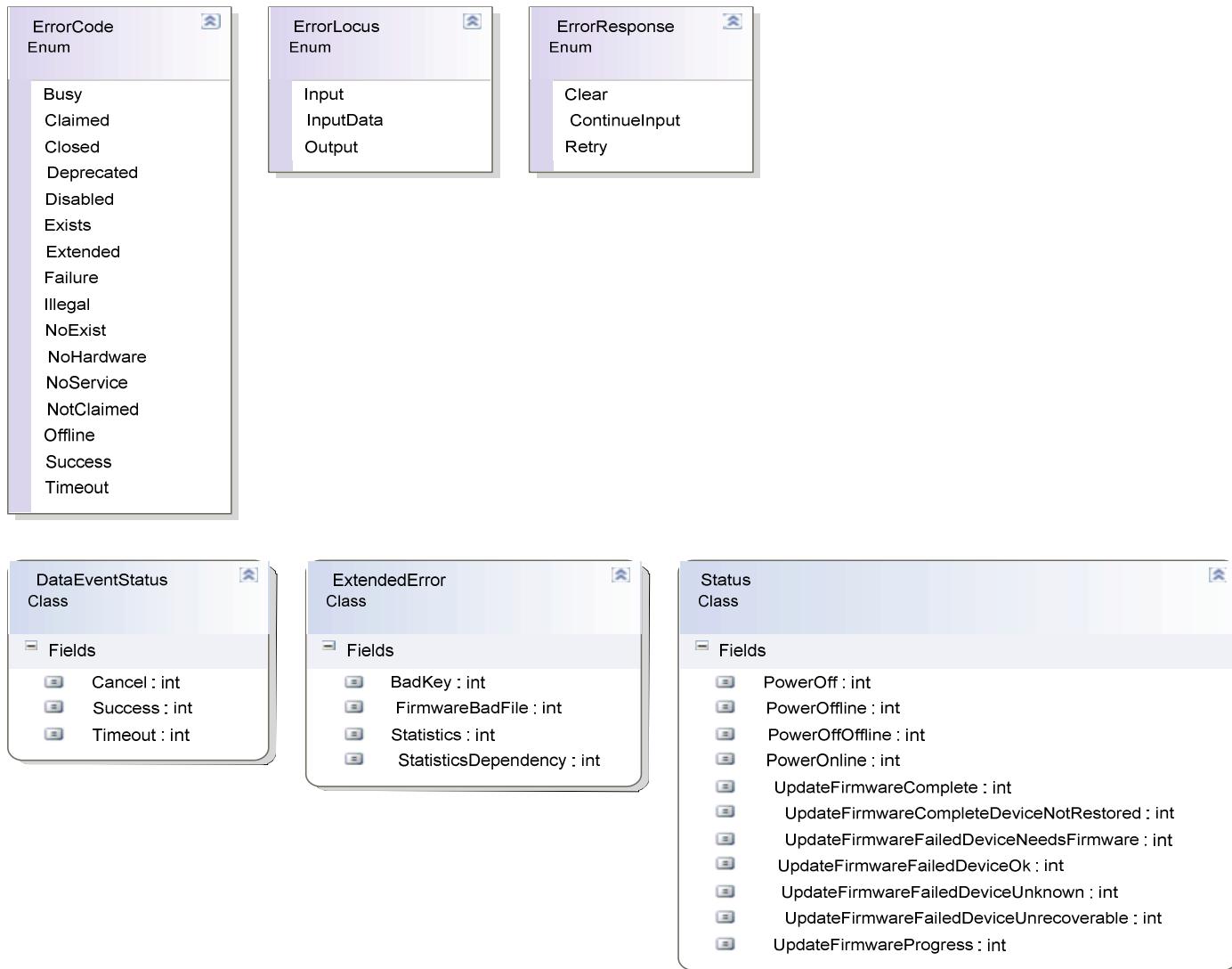
Personal Identification Number Pad (PIN Pad)



Personal Identification Number Pad (PIN Pad) Events



Personal Identification Number Pad (PIN Pad) Events



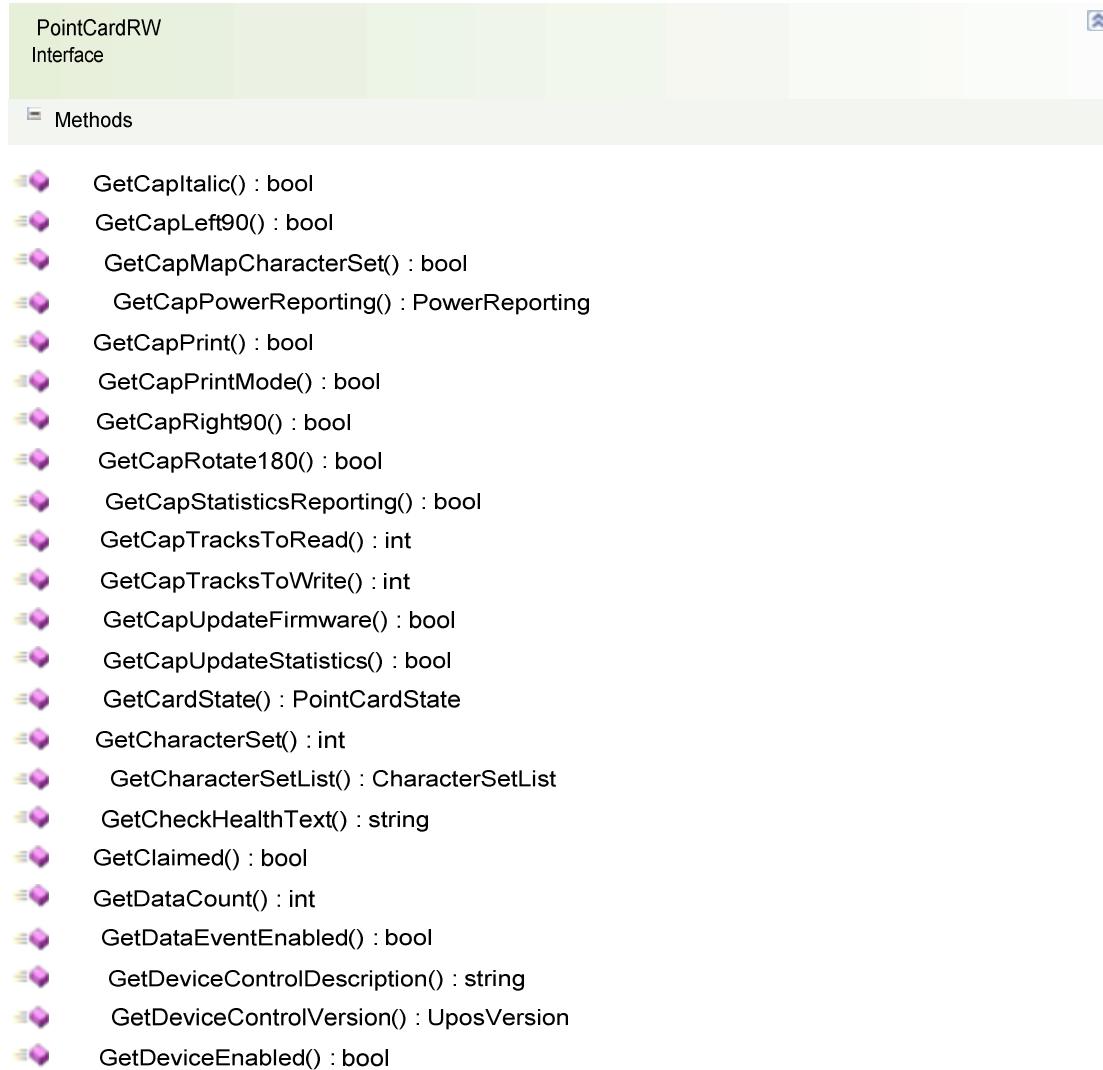
Point Card Reader/Writer (Point Card R/W)

PointCardRW
Interface

Methods

- BeginInsertion(int Timeout) : void
- BeginRemoval(int Timeout) : void
- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- CleanCard() : void
- ClearInput() : void
- ClearInputProperties() : void
- ClearOutput() : void
 - ClearPrintWrite(PointCardAreas Kind, int HPosition, int VPosition, int Width, int Height) : void
- Close(string EndpointAddress) : void
 - CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
- DirectIO(int Command, int Data, object Obj) : DirectIOData
- EndInsertion() : void
- EndRemoval() : void
- GetCapBold() : bool
- GetCapCardEntranceSensor() : bool
- GetCapCharacterSet() : int
- GetCapCleanCard() : bool
- GetCapClearPrint() : bool
- GetCapCompareFirmwareVersion() : bool
- GetCapDHigh() : bool
- GetCapDWide() : bool
- GetCapDWideDHigh() : bool

Point Card Reader/Writer (Point Card R/W)

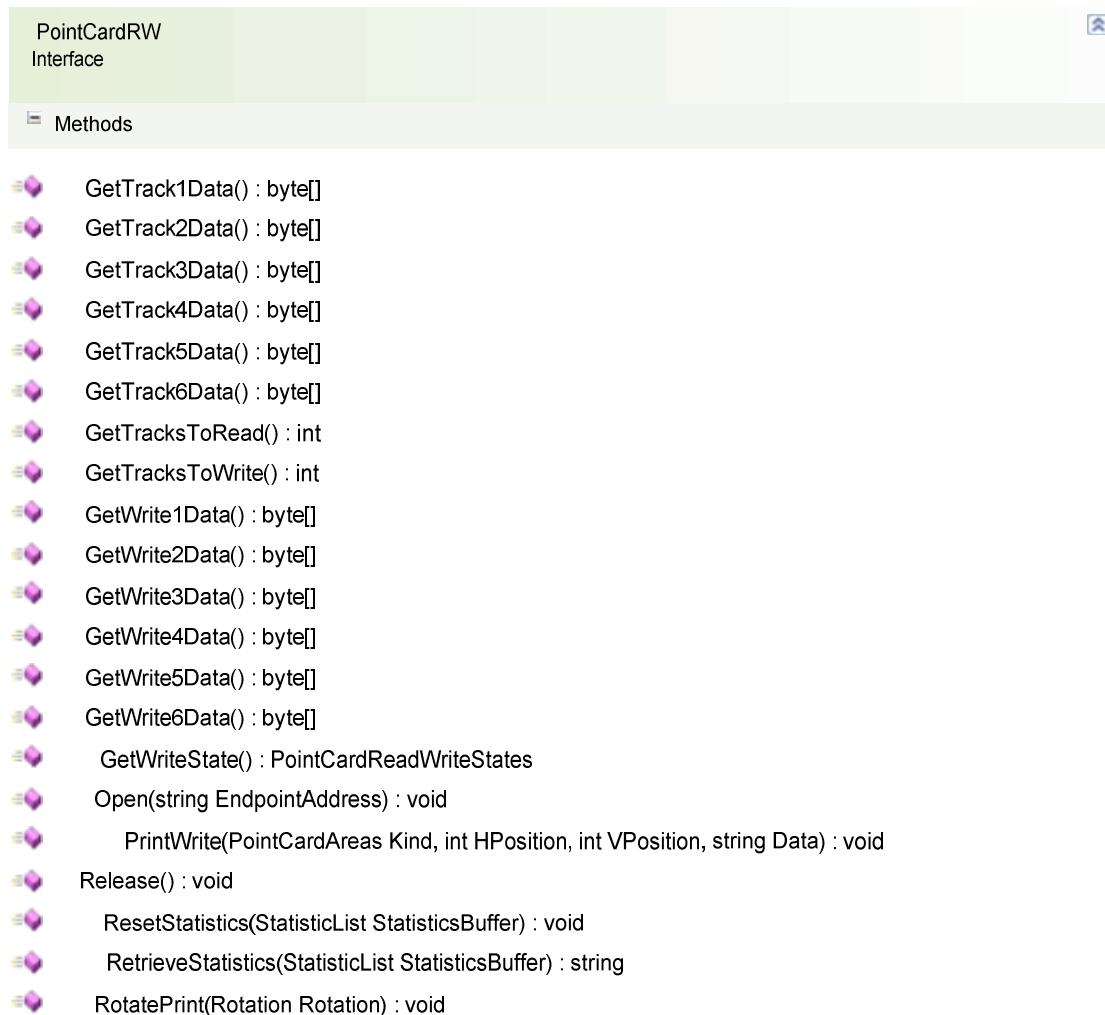


Point Card Reader/Writer (Point Card R/W)

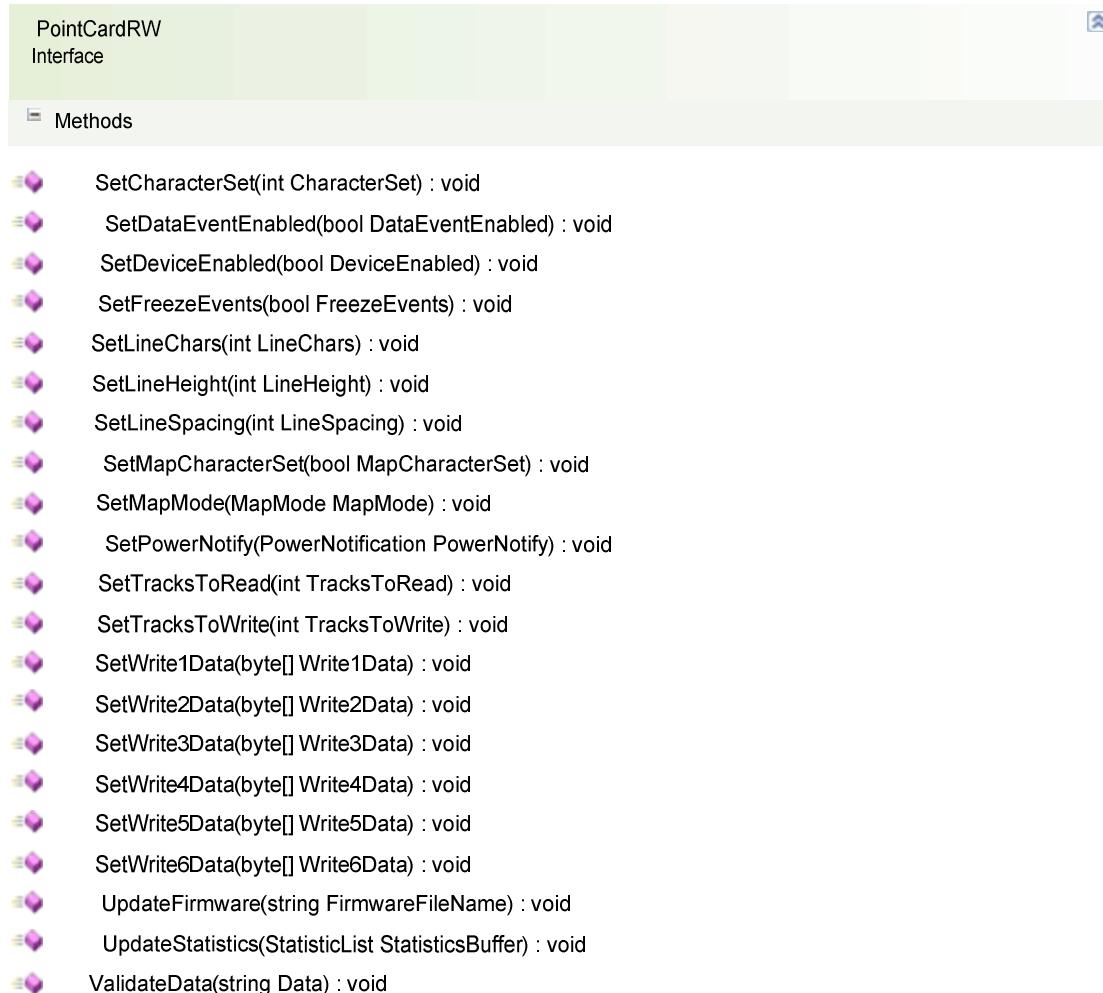
The diagram shows a UML class named "PointCardRW" with a light green background. It has a section titled "Methods" which contains a list of 25 methods, each preceded by a small purple icon. The methods are:

- GetDeviceServiceDescription() : string
- GetDeviceServiceVersion() : UposVersion
- GetFontTypefaceList() : FontTypefaceList
- GetFreezeEvents() : bool
- GetLineChars() : int
- GetLineCharsList() : LineCharsList
- GetLineHeight() : int
- GetLineSpacing() : int
- GetLineWidth() : int
- GetMapCharacterSet() : bool
- GetMapMode() : MapMode
- GetMaxLine() : int
- GetOutputID() : int
- GetPhysicalDeviceDescription() : string
- GetPhysicalDeviceName() : string
- GetPowerNotify() : PowerNotification
- GetPowerState() : PowerState
- GetPrintHeight() : int
- GetReadState() : PointCardReadWriteStates
- GetRecvLength() : PointCardReceiveLengths
- GetSidewaysMaxChars() : int
- GetSidewaysMaxLines() : int
- GetState() : ControlState

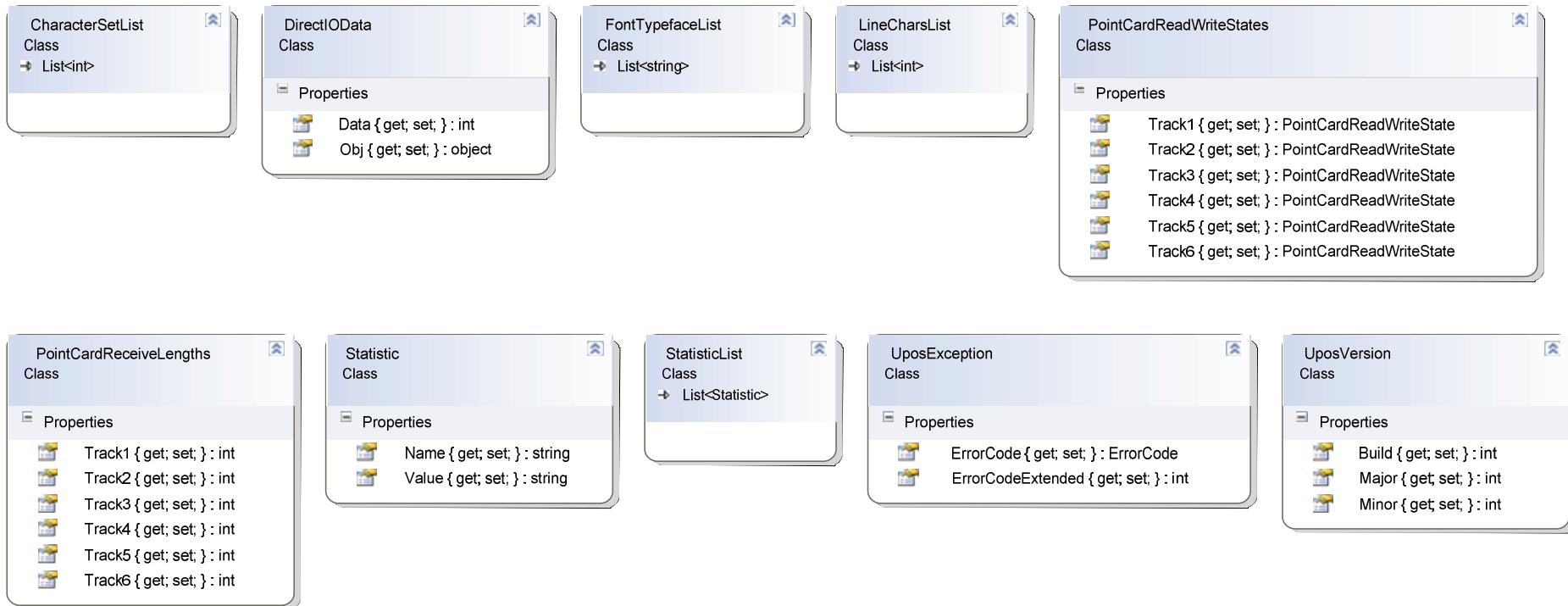
Point Card Reader/Writer (Point Card R/W)



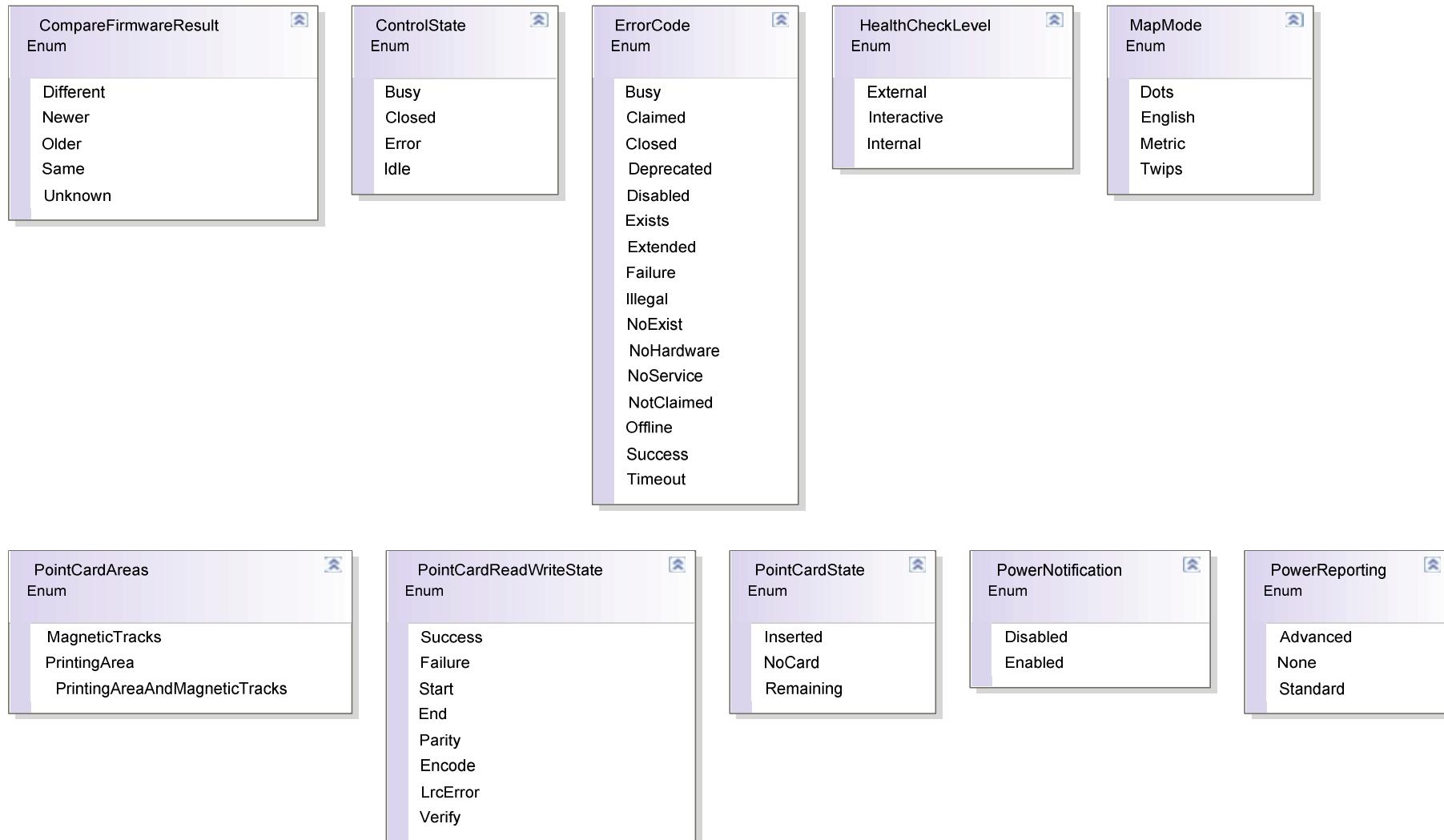
Point Card Reader/Writer (Point Card R/W)



Point Card Reader/Writer (Point Card R/W)

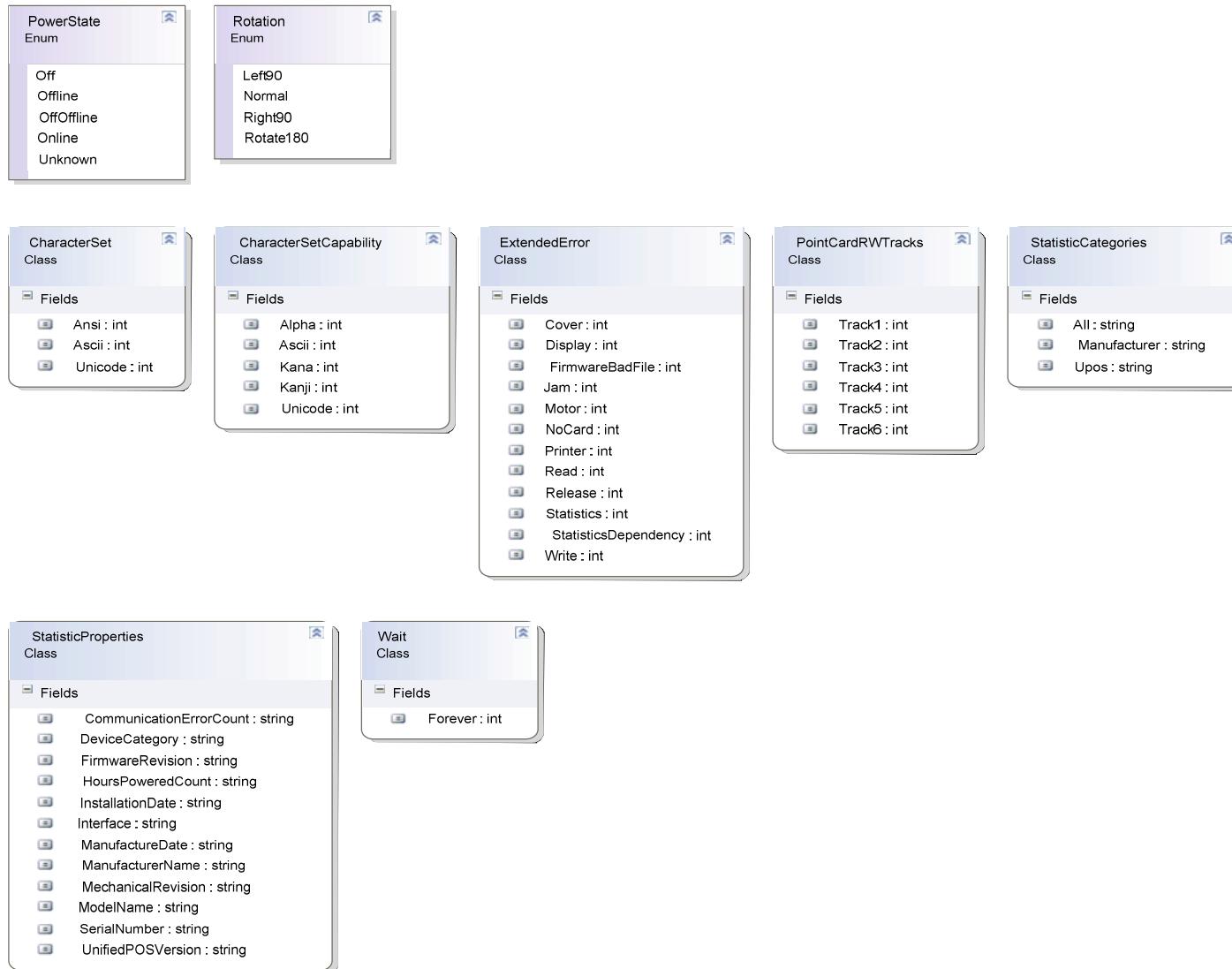


Point Card Reader/Writer (Point Card R/W)

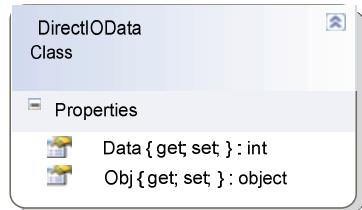
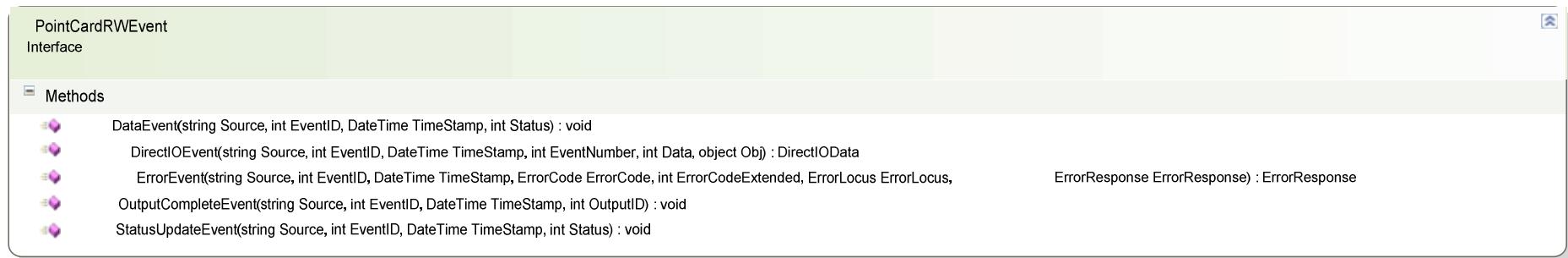


WS-POS 1.1 Technical Specification

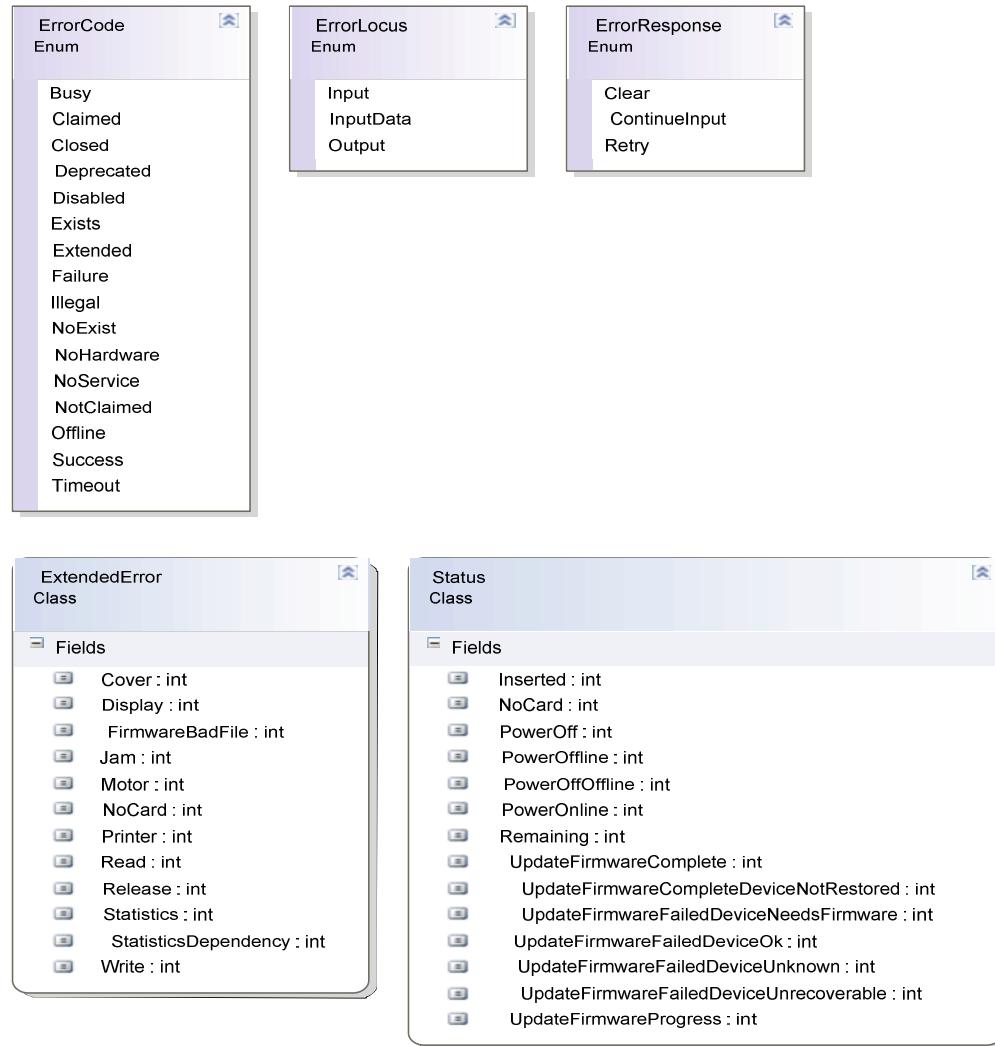
Point Card Reader/Writer (Point Card R/W)



Point Card Reader/Writer (Point Card R/W) Events



Point Card Reader/Writer (Point Card R/W) Events



POS Keyboard

POSKeyboard
Interface

Methods

- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- ClearInput() : void
- Close(string EndpointAddress) : void
 - CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
 - DirectIO(int Command, int Data, object Obj) : DirectIOData
- GetAutoDisable() : bool
- GetCapCompareFirmwareVersion() : bool
- GetCapKeyUp() : bool
- GetCapPowerReporting() : PowerReporting
- GetCapStatisticsReporting() : bool
- GetCapUpdateFirmware() : bool
- GetCapUpdateStatistics() : bool
- GetCheckHealthText() : string
- GetClaimed() : bool
- GetDataCount() : int
- GetDataEventEnabled() : bool
- GetDeviceControlDescription() : string
- GetDeviceControlVersion() : UposVersion
- GetDeviceEnabled() : bool
- GetDeviceServiceDescription() : string
- GetDeviceServiceVersion() : UposVersion
- GetEventTypes() : KeyboardEventType
- GetFreezeEvents() : bool

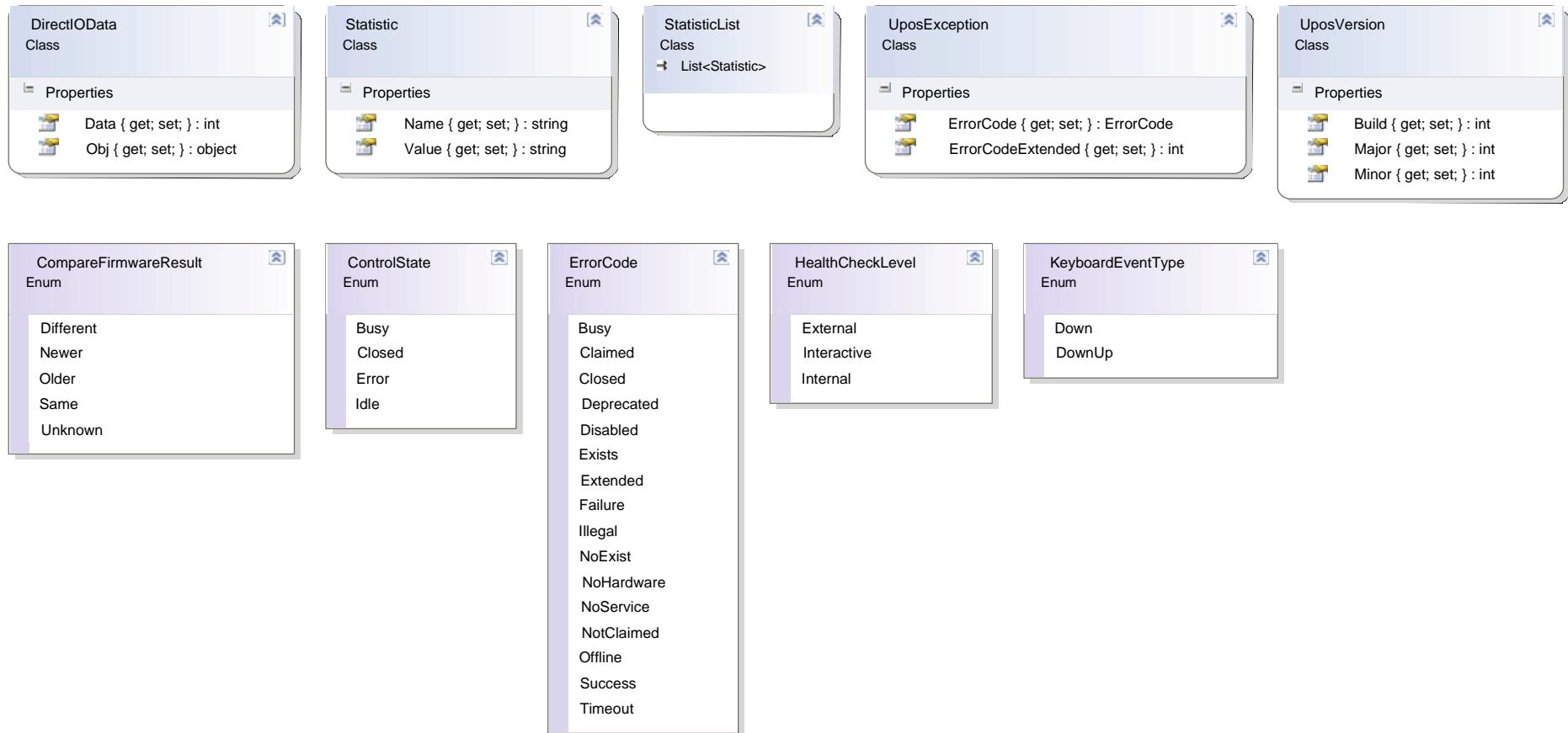
POS Keyboard



Methods

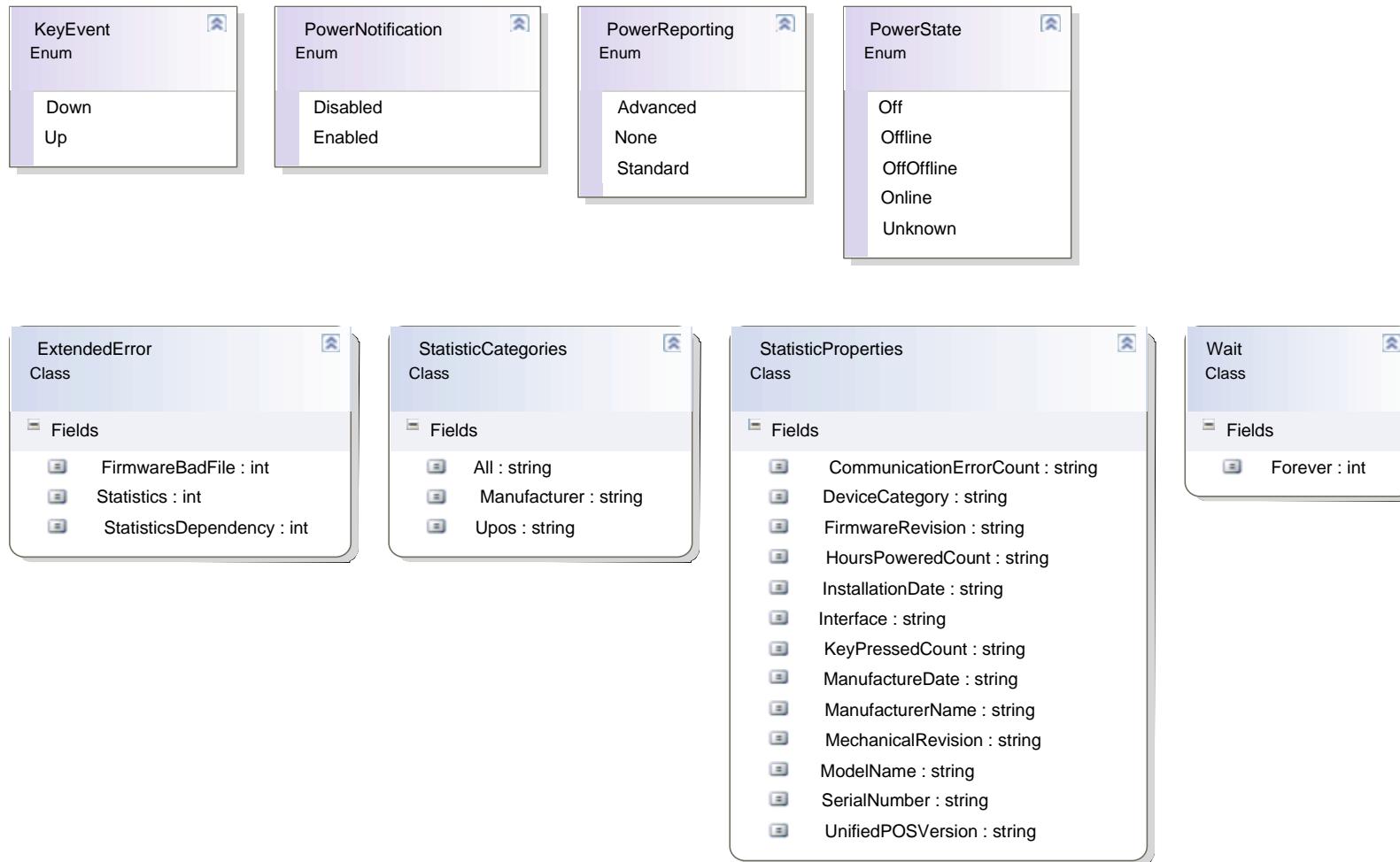
- GetPhysicalDeviceDescription() : string
- GetPhysicalDeviceName() : string
- GetPOSKeyData() : int
- GetPOSKeyEventType() : KeyEvent
- GetPowerNotify() : PowerNotification
- GetPowerState() : PowerState
- GetState() : ControlState
- Open(string EndpointAddress) : void
- Release() : void
- ResetStatistics(StatisticList StatisticsBuffer) : void
- RetrieveStatistics(StatisticList StatisticsBuffer) : string
- SetAutoDisable(bool AutoDisable) : void
- SetDataEventEnabled(bool DataEventEnabled) : void
- SetDeviceEnabled(bool DeviceEnabled) : void
- SetEventTypes(KeyboardEventType EventTypes) : void
- SetFreezeEvents(bool FreezeEvents) : void
- SetPowerNotify(PowerNotification PowerNotify) : void
- UpdateFirmware(string FirmwareFileName) : void
- UpdateStatistics(StatisticList StatisticsBuffer) : void

POS Keyboard



WS-POS 1.1 Technical Specification

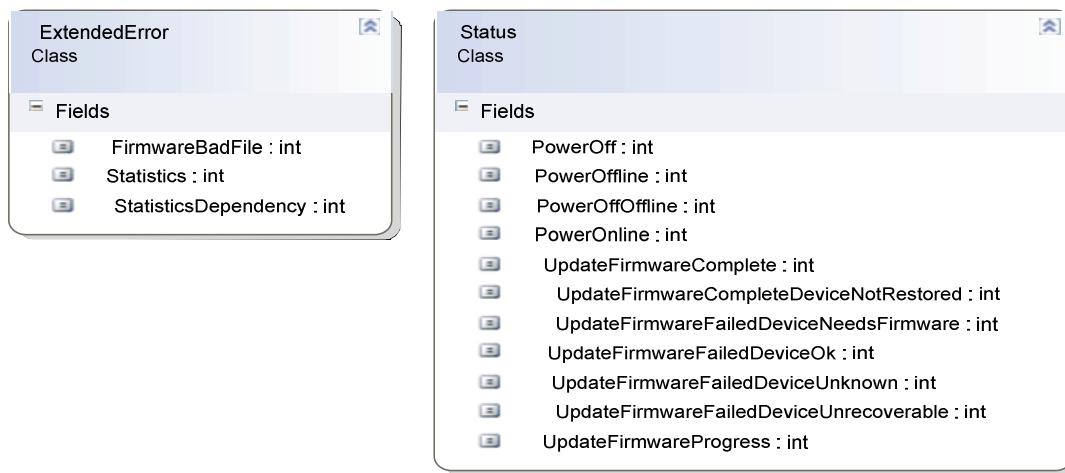
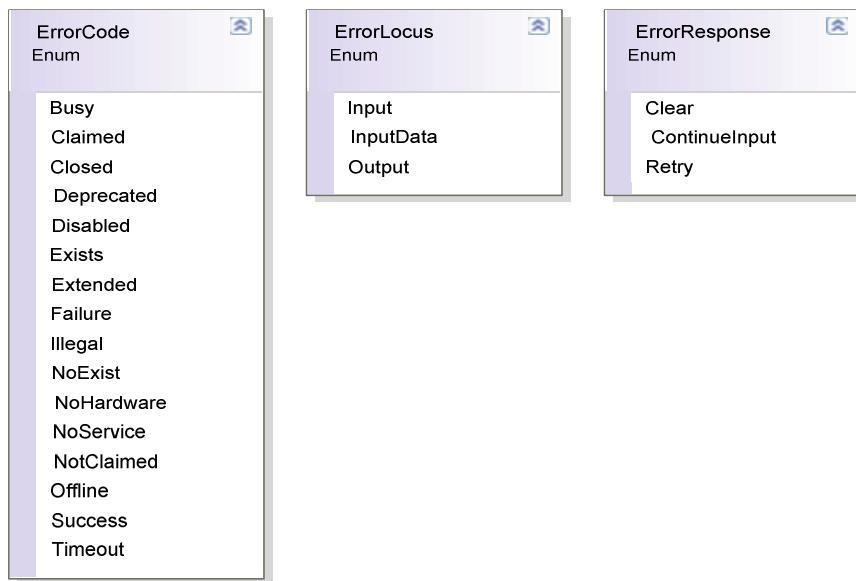
POS Keyboard



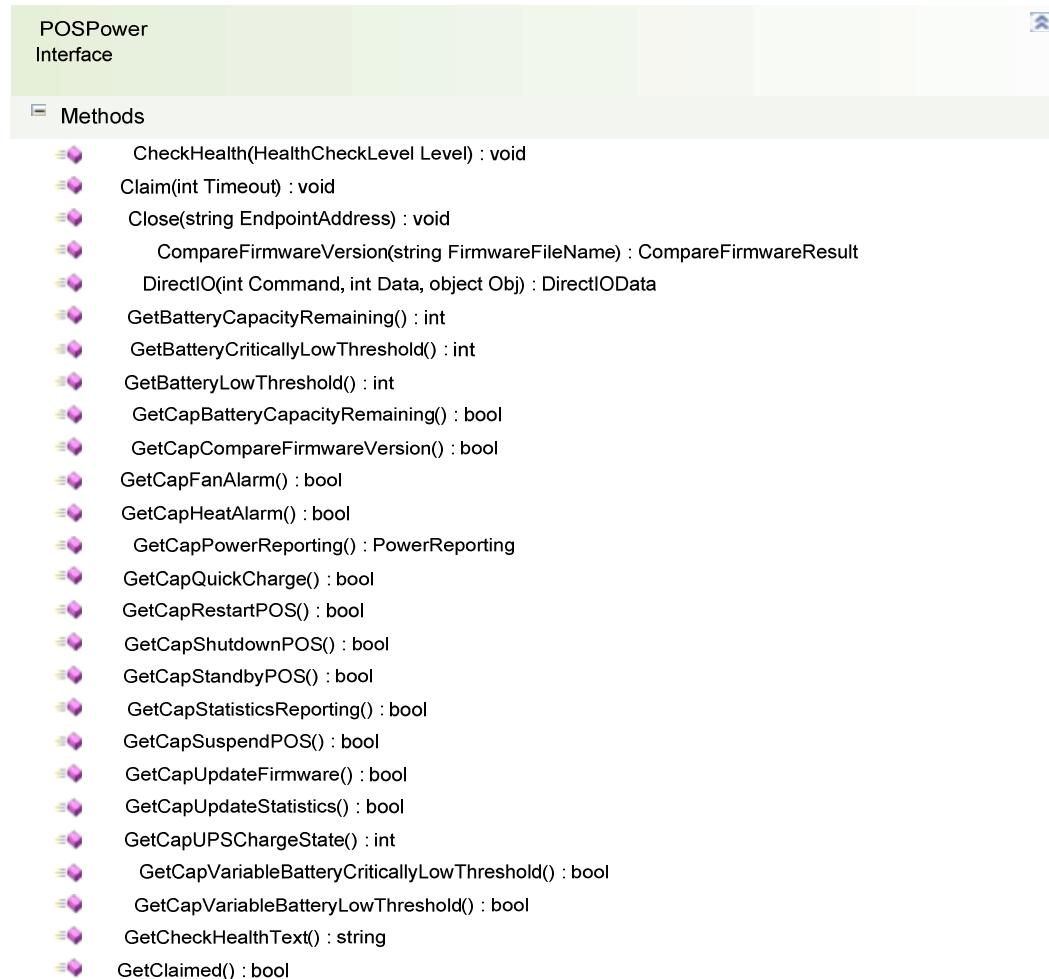
POS Keyboard Events



POS Keyboard Events



POS Power



The screenshot shows a software interface for the POSPower Interface. The left sidebar has a tree view with 'POSPower Interface' expanded, and 'Methods' is selected. The main pane displays a list of methods, each preceded by a small icon:

- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- Close(string EndpointAddress) : void
- CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
- DirectIO(int Command, int Data, object Obj) : DirectIOData
- GetBatteryCapacityRemaining() : int
- GetBatteryCriticallyLowThreshold() : int
- GetBatteryLowThreshold() : int
- GetCapBatteryCapacityRemaining() : bool
- GetCapCompareFirmwareVersion() : bool
- GetCapFanAlarm() : bool
- GetCapHeatAlarm() : bool
- GetCapPowerReporting() : PowerReporting
- GetCapQuickCharge() : bool
- GetCapRestartPOS() : bool
- GetCapShutdownPOS() : bool
- GetCapStandbyPOS() : bool
- GetCapStatisticsReporting() : bool
- GetCapSuspendPOS() : bool
- GetCapUpdateFirmware() : bool
- GetCapUpdateStatistics() : bool
- GetCapUPSChargeState() : int
- GetCapVariableBatteryCriticallyLowThreshold() : bool
- GetCapVariableBatteryLowThreshold() : bool
- GetCheckHealthText() : string
- GetClaimed() : bool

POS Power



Methods

- GetDeviceControlDescription() : string
- GetDeviceControlVersion() : UposVersion
- GetDeviceEnabled() : bool
- GetDeviceServiceDescription() : string
- GetDeviceServiceVersion() : UposVersion
- GetEnforcedShutdownDelayTime() : int
- GetFreezeEvents() : bool
- GetPhysicalDeviceDescription() : string
- GetPhysicalDeviceName() : string
- GetPowerFailDelayTime() : int
- GetPowerNotify() : PowerNotification
- GetPowerSource() : PowerSource
- GetPowerState() : PowerState
- GetQuickChargeMode() : bool
- GetQuickChargeTime() : int
- GetState() : ControlState
- GetUPSChargeState() : int
- Open(string EndpointAddress) : void
- Release() : void
- ResetStatistics(StatisticList StatisticsBuffer) : void

POS Power

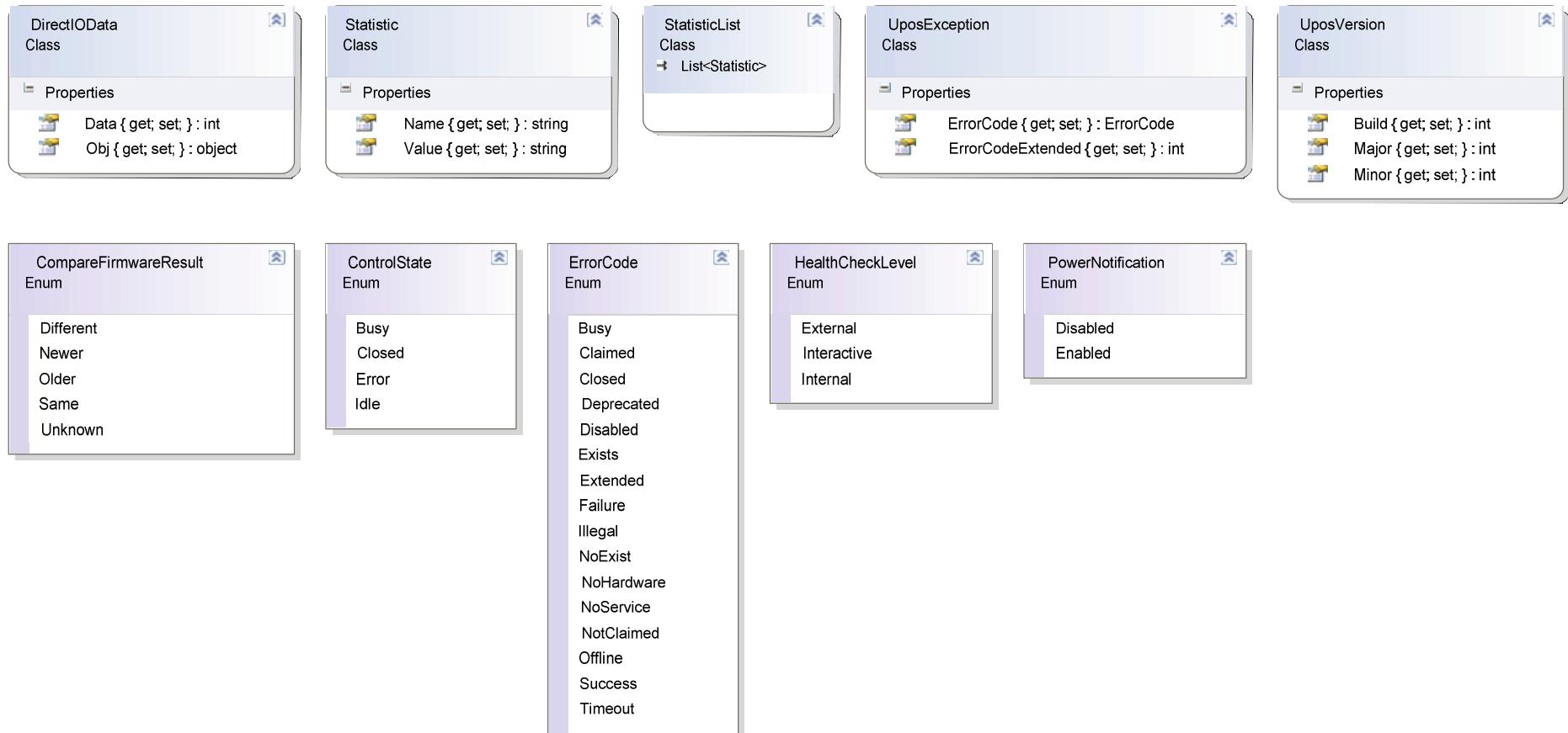


Methods

- RestartPOS() : void
- RetrieveStatistics(StatisticList StatisticsBuffer) : string
- SetBatteryCriticallyLowThreshold(int BatteryCriticallyLowThreshold) : void
- SetBatteryLowThreshold(int BatteryLowThreshold) : void
- SetDeviceEnabled(bool DeviceEnabled) : void
- SetEnforcedShutdownDelayTime(int EnforcedShutdownDelayTime) : void
- SetFreezeEvents(bool FreezeEvents) : void
- SetPowerNotify(PowerNotification PowerNotify) : void
- ShutdownPOS() : void
- StandbyPOS(SystemStateChangeReason Reason) : void
- SuspendPOS(SystemStateChangeReason Reason) : void
- UpdateFirmware(string FirmwareFileName) : void
- UpdateStatistics(StatisticList StatisticsBuffer) : void

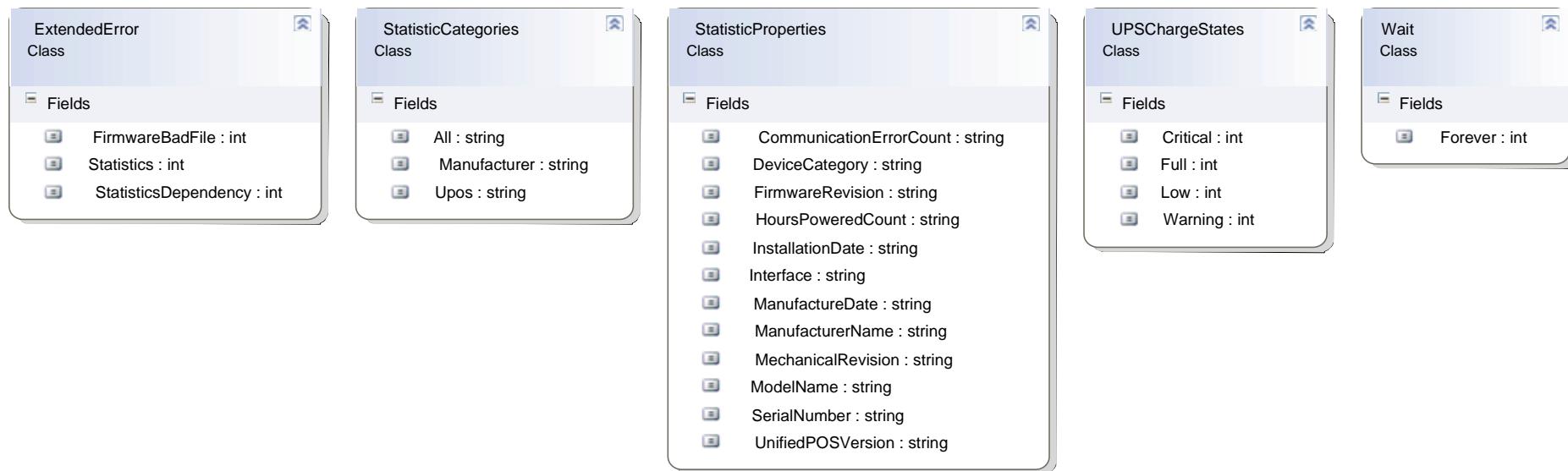
WS-POS 1.1 Technical Specification

POS Power

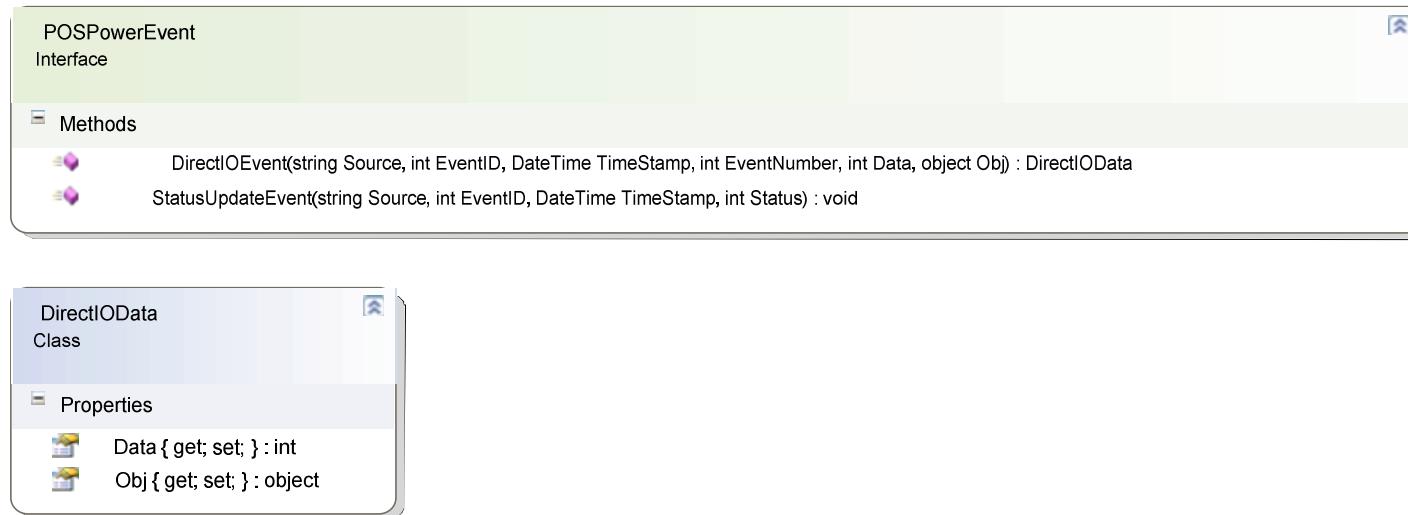


WS-POS 1.1 Technical Specification

POS Power



POS Power Events



POS Power Events

Status	Class
Fields	
BatteryCapacityRemaining : int	
BatteryCritical : int	
BatteryLow : int	
FanRunning : int	
FanStopped : int	
PowerOff : int	
PowerOffline : int	
PowerOffOffline : int	
PowerOnline : int	
PowerSource : int	
Restart : int	
ShutDown : int	
Standby : int	
Suspend : int	
TemperatureHigh : int	
TemperatureOK : int	
UpdateFirmwareComplete : int	
UpdateFirmwareCompleteDeviceNotRestored : int	
UpdateFirmwareFailedDeviceNeedsFirmware : int	
UpdateFirmwareFailedDeviceOk : int	
UpdateFirmwareFailedDeviceUnknown : int	
UpdateFirmwareFailedDeviceUnrecoverable : int	
UpdateFirmwareProgress : int	
UpsCritical : int	
UpsFull : int	
UpsLow : int	
UpsWarning : int	
UserStandby : int	
UserSuspend : int	

POS Printer

The diagram shows a UML class named **POSPrinter Interface**. It has a single association named **Methods** pointing to a list of method signatures. The methods listed are:

- BeginInsertion(int Timeout) : void
- BeginRemoval(int Timeout) : void
- ChangePrintSide(PrinterSide Side) : void
- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- ClearOutput() : void
- ClearPrintArea() : void
- Close(string EndpointAddress) : void
 - CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
- CutPaper(int Percentage) : void
- DirectIO(int Command, int Data, object Obj) : DirectIOData
- DrawRuledLine(PrinterStation Station, PositionList PositionList, int LineDirection, int LineWidth, PrinterLineStyle LineStyle , int LineColor) : void
- EndInsertion() : void
- EndRemoval() : void
- GetAsyncMode() : bool
- GetCapCharacterSet() : CharacterSetCapability
- GetCapCompareFirmwareVersion() : bool
- GetCapConcurrentJrnRec() : bool
- GetCapConcurrentJrnSlp() : bool
- GetCapConcurrentPageMode() : bool
- GetCapConcurrentRecSlp() : bool
- GetCapCoverSensor() : bool
- GetCapJrn2Color() : bool
- GetCapJrnBold() : bool
- GetCapJrnCartridgeSensor() : int

WS-POS 1.1 Technical Specification

POS Printer



The diagram shows a UML class named "POSPrinter Interface". It has a single association labeled "Methods" pointing to a list of methods. The methods listed are:

- GetCapJrnColor() : int
- GetCapJrnDhigh() : bool
- GetCapJrnDwide() : bool
- GetCapJrnDwideDhigh() : bool
- GetCapJrnEmptySensor() : bool
- GetCapJrnItalic() : bool
- GetCapJrnNearEndSensor() : bool
- GetCapJrnPresent() : bool
- GetCapJrnUnderline() : bool
- GetCapMapCharacterSet() : bool
- GetCapPowerReporting() : PowerReporting
- GetCapRec2Color() : bool
- GetCapRecBarcode() : bool
- GetCapRecBitmap() : bool
- GetCapRecBold() : bool
- GetCapRecCartridgeSensor() : int
- GetCapRecColor() : int
- GetCapRecDhigh() : bool
- GetCapRecDwide() : bool
- GetCapRecDwideDhigh() : bool
- GetCapRecEmptySensor() : bool
- GetCapRecItalic() : bool
- GetCapRecLeft90() : bool

POS Printer

POSPrinter
Interface

Methods

- GetCapRecMarkFeed() : int
- GetCapRecNearEndSensor() : bool
- GetCapRecPageMode() : bool
- GetCapRecPapercut() : bool
- GetCapRecPresent() : bool
- GetCapRecRight90() : bool
- GetCapRecRotate180() : bool
- GetCapRecRuledLine() : int
- GetCapRecStamp() : bool
- GetCapRecUnderline() : bool
- GetCapSlip2Color() : bool
- GetCapSlipBarcode() : bool
- GetCapSlipBitmap() : bool
- GetCapSlipBold() : bool
- GetCapSlipBothSidesPrint() : bool
- GetCapSlipCartridgeSensor() : int
- GetCapSlipColor() : int
- GetCapSlipDhigh() : bool

POS Printer

The diagram shows a UML class named "POSPrinter Interface". It has a single section labeled "Methods" which contains the following list of methods:

- GetCapSlipDwide() : bool
- GetCapSlipDwideDhigh() : bool
- GetCapSlipEmptySensor() : bool
- GetCapSlipFullslip() : bool
- GetCapSlipItalic() : bool
- GetCapSlipLeft90() : bool
- GetCapSlipNearEndSensor() : bool
- GetCapSlipPageMode() : bool
- GetCapSlipPresent() : bool
- GetCapSlipRight90() : bool
- GetCapSlipRotate180() : bool
- GetCapSlipRuledLine() : int
- GetCapSlipUnderline() : bool
- GetCapStatisticsReporting() : bool
- GetCapTransaction() : bool
- GetCapUpdateFirmware() : bool
- GetCapUpdateStatistics() : bool
- GetCartridgeNotify() : PrinterCartridgeNotify
- GetCharacterSet() : int
- GetCharacterSetList() : CharacterSetList
- GetCheckHealthText() : string
- GetClaimed() : bool
- GetCoverOpen() : bool

POS Printer



The diagram shows a UML class named "POSPrinter Interface". It has a single section labeled "Methods" which contains a list of methods. Each method is preceded by a small icon of a person with a purple circle on their head.

- = GetDeviceControlDescription() : string
- = GetDeviceControlVersion() : UposVersion
- = GetDeviceEnabled() : bool
- = GetDeviceServiceDescription() : string
- = GetDeviceServiceVersion() : UposVersion
- = GetErrorLevel() : PrinterErrorLevel
- = GetErrorStation() : PrinterStation
- = GetErrorString() : string
- = GetFlagWhenIdle() : bool
- = GetFontTypefaceList() : FontTypefaceList
- = GetFreezeEvents() : bool
 - = GetJrnCartridgeState() : PrinterCartridgeStates
 - = GetJrnCurrentCartridge() : int
 - = GetJrnEmpty() : bool
 - = GetJrnLetterQuality() : bool
 - = GetJrnLineChars() : int
 - = GetJrnLineCharsList() : LineCharsList
 - = GetJrnLineHeight() : int
 - = GetJrnLineSpacing() : int
 - = GetJrnLineWidth() : int
 - = GetJrnNearEnd() : bool
 - = GetMapCharacterSet() : bool
 - = GetMapMode() : MapMode
 - = GetOutputID() : int
 - = GetPageModeArea() : Point
 - = GetPageModeDescriptor() : int
 - = GetPageModeHorizontalPosition() : int
 - = GetPageModePrintArea() : Rectangle

POS Printer

The diagram shows a UML class named "POSPrinter Interface". It has a single section labeled "Methods" which contains a list of 34 methods, each preceded by a small purple icon.

- GetPageModePrintDirection() : PageModePrintDirection
- GetPageModeStation() : PrinterStation
- GetPageModeVerticalPosition() : int
- GetPhysicalDeviceDescription() : string
- GetPhysicalDeviceName() : string
- GetPowerNotify() : PowerNotification
- GetPowerState() : PowerState
- GetRecBarcodeRotationList() : RotationList
- GetRecBitmapRotationList() : RotationList
- GetRecCartridgeState() : PrinterCartridgeStates
- GetRecCurrentCartridge() : int
- GetRecEmpty() : bool
- GetRecLetterQuality() : bool
- GetRecLineChars() : int
- GetRecLineCharsList() : LineCharsList
- GetRecLineHeight() : int
- GetRecLineSpacing() : int
- GetRecLinesToPaperCut() : int
- GetRecLineWidth() : int
- GetRecNearEnd() : bool
- GetRecSidewaysMaxChars() : int
- GetRecSidewaysMaxLines() : int
- GetRotateSpecial() : Rotation
- GetSlipBarcodeRotationList() : RotationList
- GetSlipBitmapRotationList() : RotationList
- GetSlipCartridgeState() : PrinterCartridgeStates
- GetSlipCurrentCartridge() : int
- GetSlipEmpty() : bool
- GetSlipLetterQuality() : bool
- GetSlipLineChars() : int

POS Printer

The diagram shows a UML class named 'POSPrinter Interface' with a light green background. It has a section titled 'Methods' which contains the following list of methods:

- GetSlpLineChars() : int
- GetSlpLineCharsList() : LineCharsList
- GetSlpLineHeight() : int
- GetSlpLinesNearEndToEnd() : int
- GetSlpLineSpacing() : int
- GetSlpLineWidth() : int
- GetSlpMaxLines() : int
- GetSlpNearEnd() : bool
- GetSlpPrintSide() : PrinterSide
- GetSlpSidewaysMaxChars() : int
- GetSlpSidewaysMaxLines() : int
- GetState() : ControlState
- MarkFeed(int Type) : void
- Open(string EndpointAddress) : void
- PageModePrint(PageModePrintControl Control) : void
 - PrintBarcode(PrinterStation Station, string Data, int Symbology, int Height, int Width, int Alignment, BarCodeTextPosition TextPosition) : void
- PrintBitmap(PrinterStation Station, string FileName, int Width, int Alignment) : void
- PrintImmediate(PrinterStation Station, string Data) : void
 - PrintMemoryBitmap(PrinterStation Station, byte[] Data, BitmapType Type, int Width, int Alignment) : void
- PrintNormal(PrinterStation Station, string Data) : void
- PrintTwoNormal(PrinterStation Station, string Data1, string Data2) : void
- Release() : void
- ResetStatistics(StatisticList StatisticsBuffer) : void
- RetrieveStatistics(StatisticList StatisticsBuffer) : string
- RotatePrint(PrinterStation Station, int Rotation) : void
- SetAsyncMode(bool AsyncMode) : void
- SetBitmap(int BitmapNumber, PrinterStation Station, string FileName, int Width, int Alignment) : void

WS-POS 1.1 Technical Specification

POS Printer

The diagram shows a UML class named 'POSPrinter Interface'. It has a single section labeled 'Methods' which contains 24 method definitions, each preceded by a small icon.

- SetCartridgeNotify(PrinterCartridgeNotify CartridgeNotify) : void
- SetCharacterSet(int CharacterSet) : void
- SetDeviceEnabled(bool DeviceEnabled) : void
- SetFlagWhenIdle(bool FlagWhenIdle) : void
- SetFreezeEvents(bool FreezeEvents) : void
- SetJrnCurrentCartridge(int JrnCurrentCartridge) : void
- SetJrnLetterQuality(bool JrnLetterQuality) : void
- SetJrnLineChars(int JrnLineChars) : void
- SetJrnLineHeight(int JrnLineHeight) : void
- SetJrnLineSpacing(int JrnLineSpacing) : void
- SetLogo(PrinterLogoLocation Location, string Data) : void
- SetMapCharacterSet(bool MapCharacterSet) : void
- SetMapMode(MapMode MapMode) : void
 - SetPageModeHorizontalPosition(int PageModeHorizontalPosition) : void
 - SetPageModePrintArea(Rectangle PageModePrintArea) : void
 - SetPageModePrintDirection(PageModePrintDirection PageModePrintDirection) : void
- SetPageModeStation(PrinterStation PageModeStation) : void
- SetPageModeVerticalPosition(int PageModeVerticalPosition) : void
- SetPowerNotify(PowerNotification PowerNotify) : void
- SetRecCurrentCartridge(int RecCurrentCartridge) : void
- SetRecLetterQuality(bool RecLetterQuality) : void
- SetRecLineChars(int RecLineChars) : void
- SetRecLineHeight(int RecLineHeight) : void

POS Printer

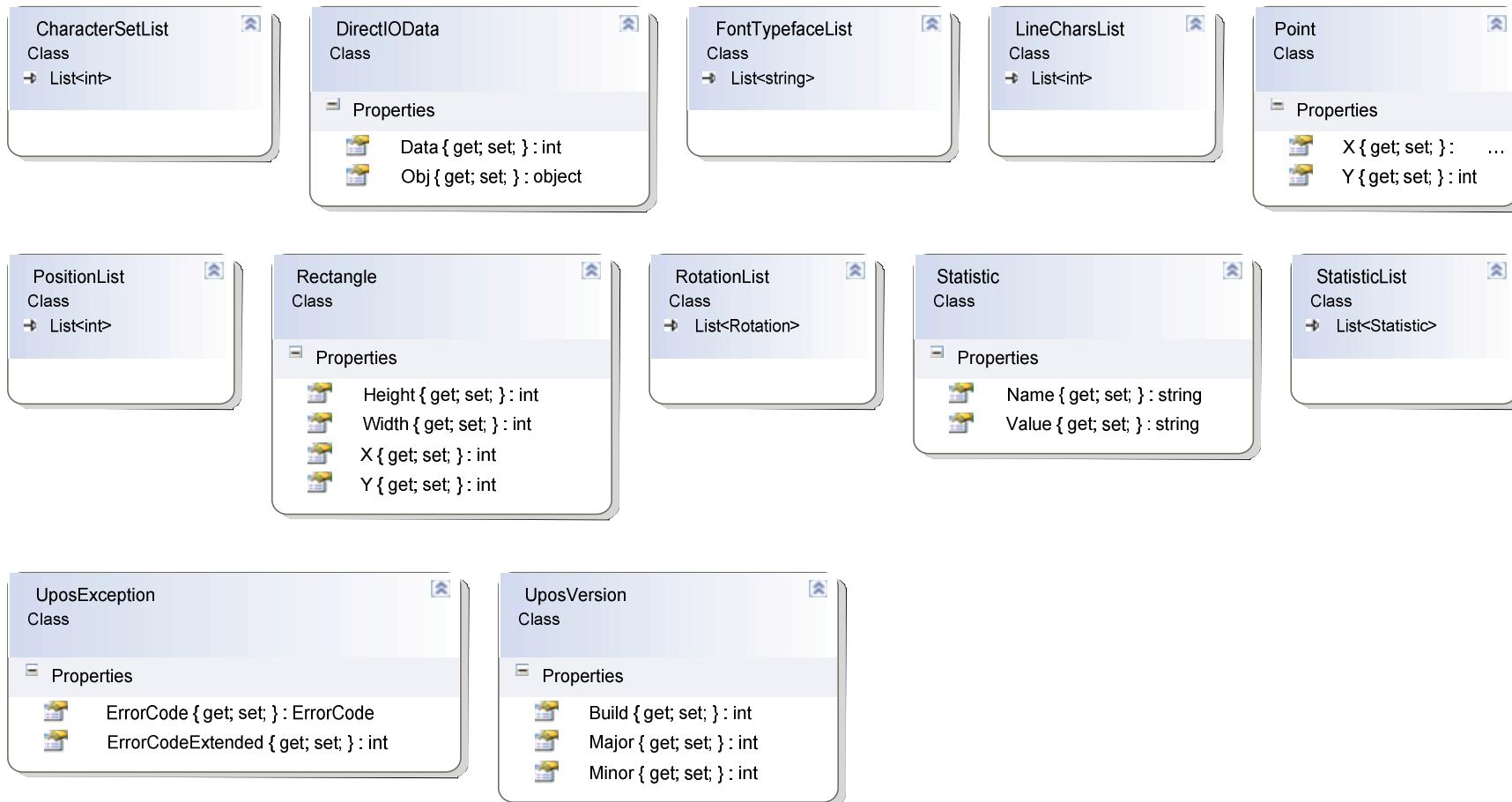
POSPrinter
Interface

Methods

- SetRecLineSpacing(int RecLineSpacing) : void
- SetRotateSpecial(Rotation RotateSpecial) : void
- SetSlpCurrentCartridge(int SlpCurrentCartridge) : void
- SetSlpLetterQuality(bool SlpLetterQuality) : void
- SetSlpLineChars(int SlpLineChars) : void
- SetSlpLineHeight(int SlpLineHeight) : void
- SetSlpLineSpacing(int SlpLineSpacing) : void
- TransactionPrint(PrinterStation Station, PrinterTransactionControl Control) : void
- UpdateFirmware(string FirmwareFileName) : void
- UpdateStatistics(StatisticList StatisticsBuffer) : void
- ValidateData(PrinterStation Station, string Data) : void

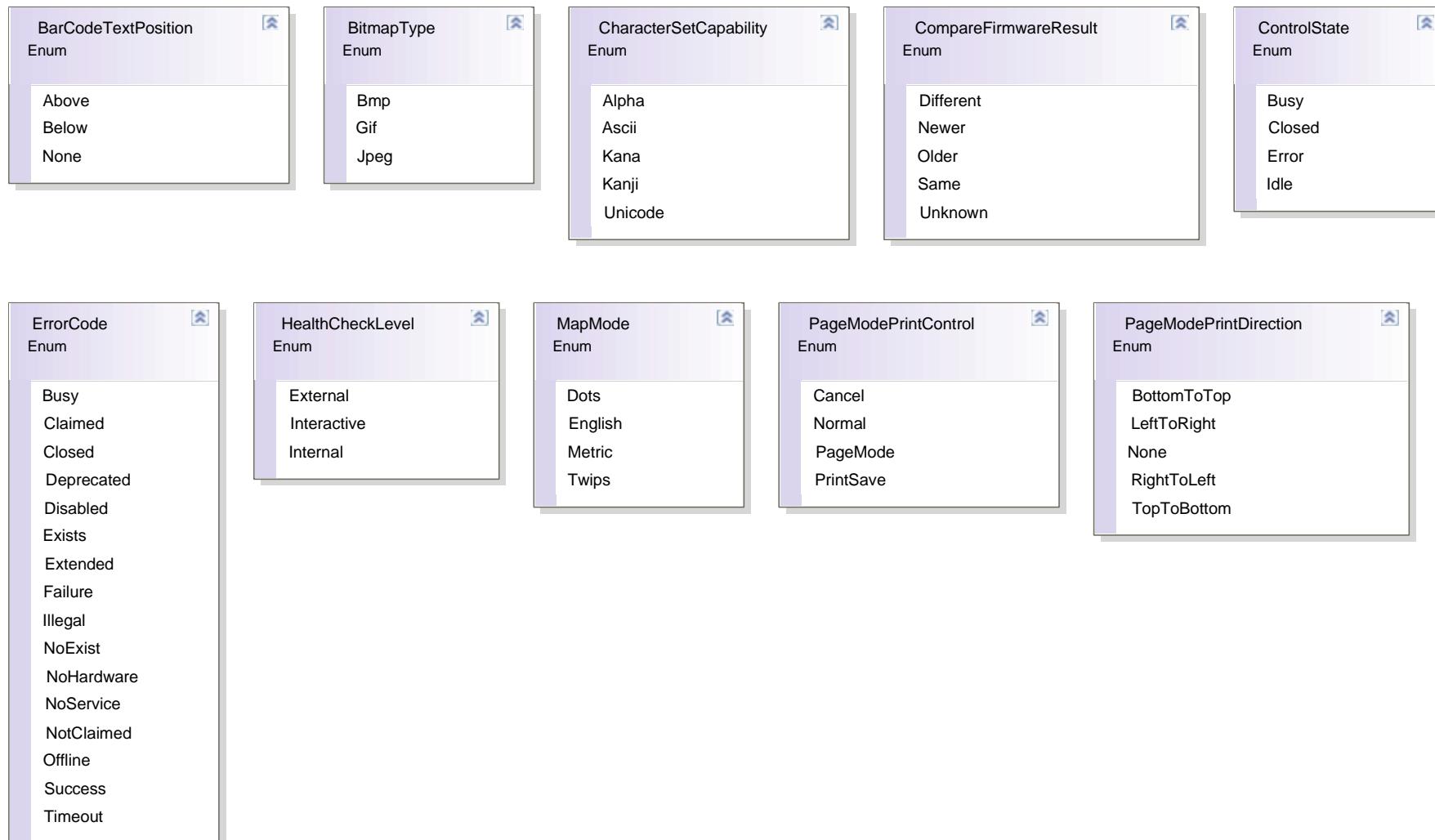
WS-POS 1.1 Technical Specification

POS Printer



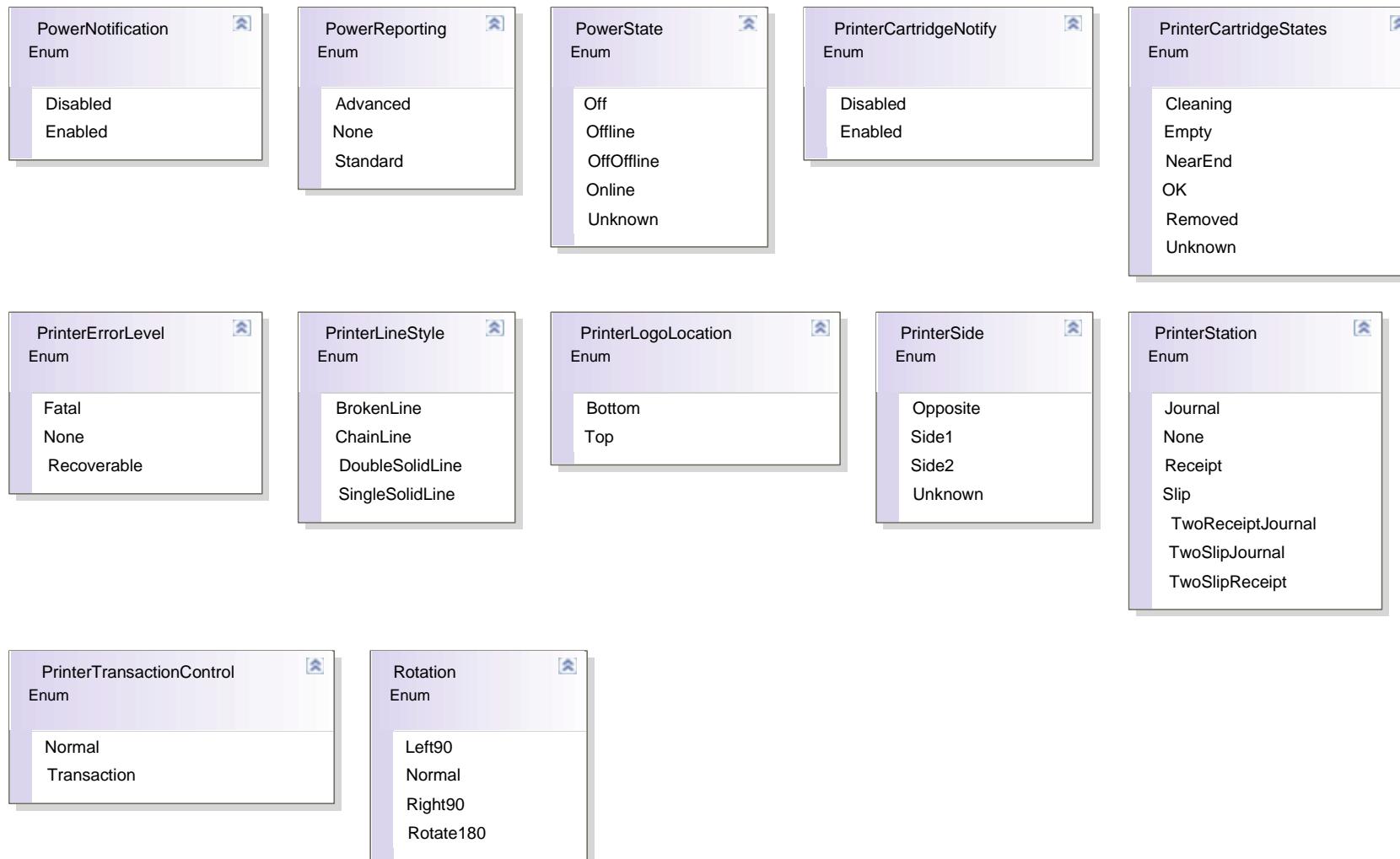
WS-POS 1.1 Technical Specification

POS Printer



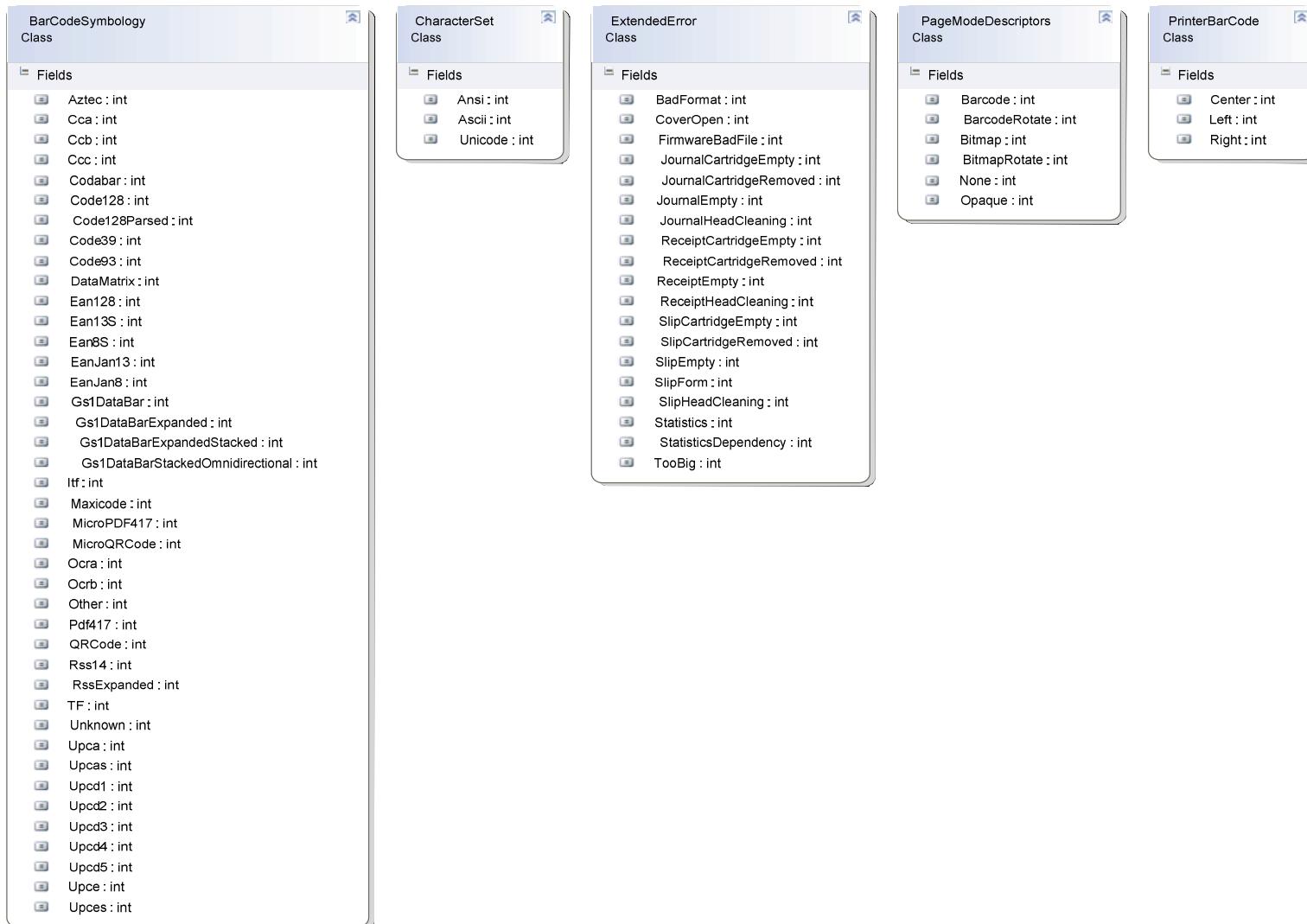
WS-POS 1.1 Technical Specification

POS Printer



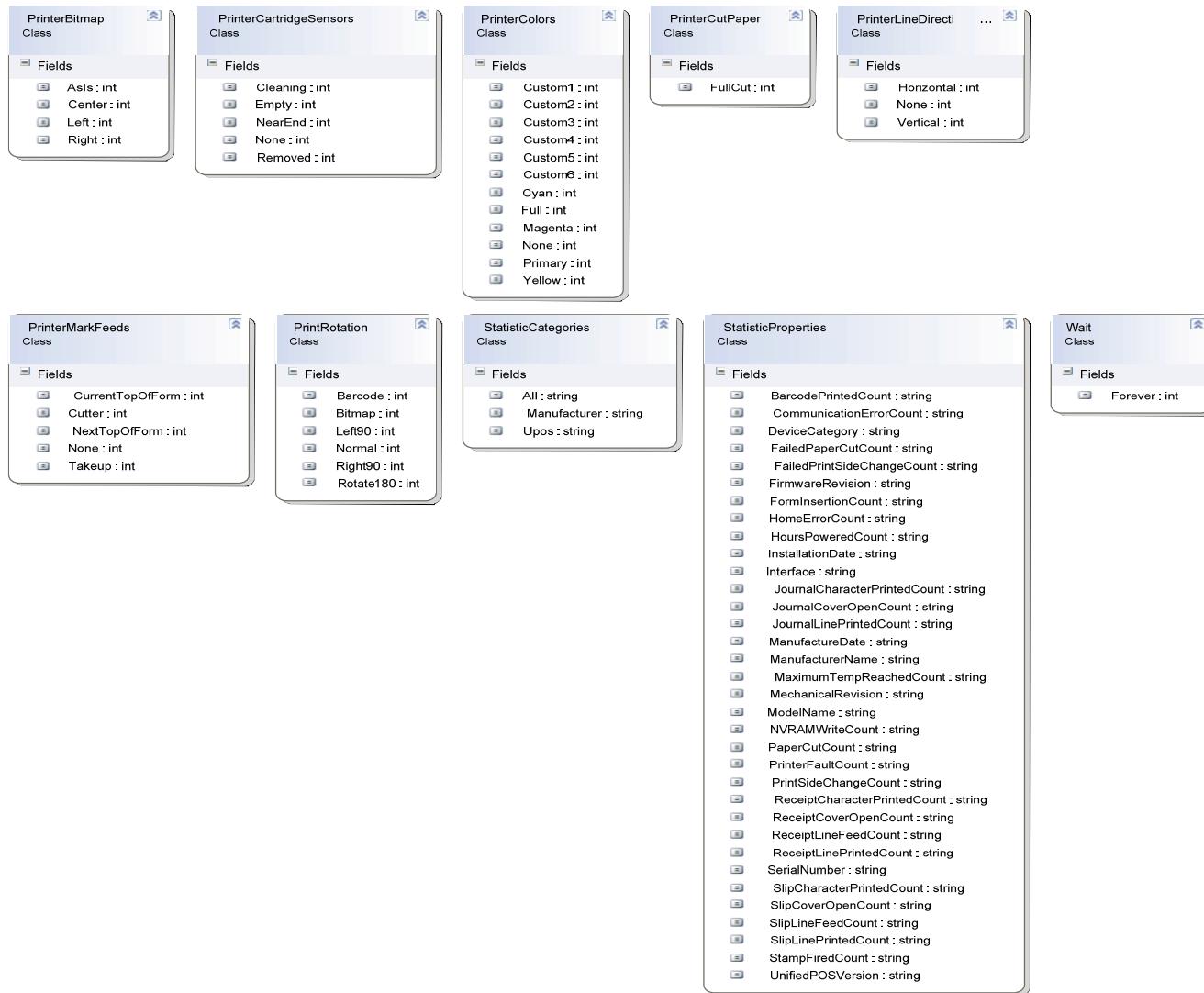
WS-POS 1.1 Technical Specification

POS Printer

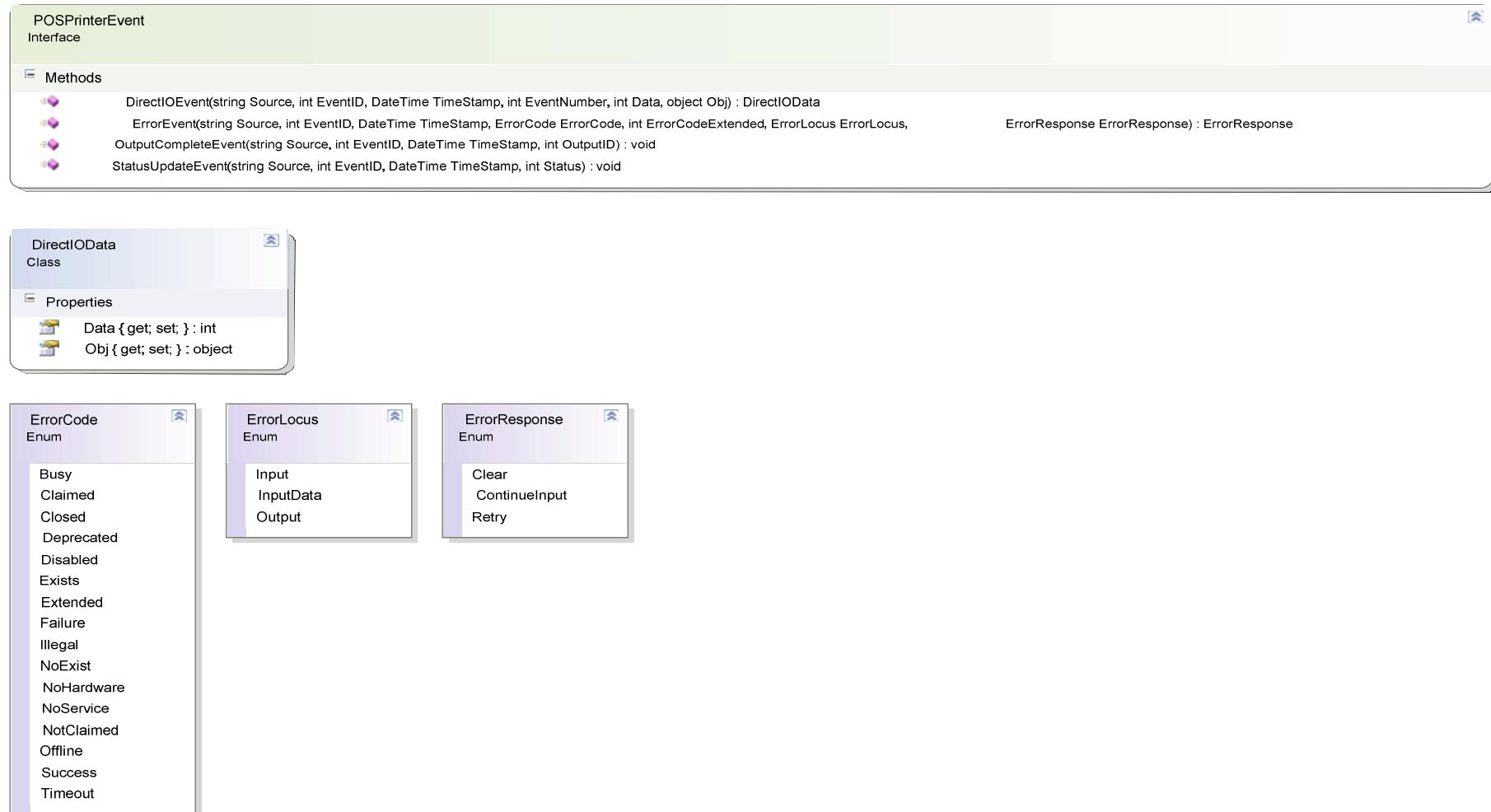


WS-POS 1.1 Technical Specification

POS Printer



POS Printer Events



POS Printer Events

ExtendedError Class	Status Class
Fields	
BadFormat : int CoverOpen : int FirmwareBadFile : int JournalCartridgeEmpty : int JournalCartridgeRemoved : int JournalEmpty : int JournalHeadCleaning : int ReceiptCartridgeEmpty : int ReceiptCartridgeRemoved : int ReceiptEmpty : int ReceiptHeadCleaning : int SlipCartridgeEmpty : int SlipCartridgeRemoved : int SlipEmpty : int SlipForm : int SlipHeadCleaning : int Statistics : int StatisticsDependency : int TooBig : int	CoverOK : int CoverOpen : int Idle : int JournalCartridgeEmpty : int JournalCartridgeNearEmpty : int JournalCartridgeOK : int JournalCoverOK : int JournalCoverOpen : int JournalEmpty : int JournalHeadCleaning : int JournalNearEmpty : int JournalPaperOK : int PowerOff : int PowerOffline : int PowerOffline : int PowerOnline : int ReceiptCartridgeEmpty : int ReceiptCartridgeNearEmpty : int ReceiptCartridgeOK : int ReceiptCoverOK : int ReceiptCoverOpen : int ReceiptEmpty : int ReceiptHeadCleaning : int ReceiptNearEmpty : int ReceiptPaperOK : int SlipCartridgeEmpty : int SlipCartridgeNearEmpty : int SlipCartridgeOK : int SlipCoverOK : int SlipCoverOpen : int SlipEmpty : int SlipHeadCleaning : int SlipNearEmpty : int SlipPaperOK : int UpdateFirmwareComplete : int UpdateFirmwareCompleteDeviceNotRestored : int UpdateFirmwareFailedDeviceNeedsFirmware : int UpdateFirmwareFailedDeviceOk : int UpdateFirmwareFailedDeviceUnknown : int UpdateFirmwareFailedDeviceUnrecoverable : int UpdateFirmwareProgress : int

Remote Order Display

RemoteOrderDisplay
Interface

Methods

- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- ClearInput() : void
- ClearOutput() : void
- ClearVideo(int Units, int Attribute) : void
- ClearVideoRegion(int Row, int Column, int Height, int Width, int Attribute) : void
- Close(string EndpointAddress) : void
- CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
- ControlClock(int Units, ClockFunction ClockFunction, int ClockId, int Hours, int Minutes, int Seconds, int Row, int Column, int Attribute, ClockMode Mode) : void
- ControlCursor(int Units, VideoCursorType CursorType) : void
- CopyVideoRegion(int Units, int Row, int Column, int Height, int Width, int TargetRow, int TargetColumn) : void
- DirectIO(int Command, int Data, object Obj) : DirectIOData
- DisplayData(int Units, int Row, int Column, int Attribute, string Data) : void
- DrawBox(int Units, int Row, int Column, int Height, int Width, int Attribute, BorderType BorderType) : void
- FreeVideoRegion(int Units, int BufferId) : void
- GetAsyncMode() : bool
- GetAutoToneDuration() : int
- GetAutoToneFrequency() : int
- GetCapCompareFirmwareVersion() : bool
- GetCapMapCharacterSet() : bool
- GetCapPowerReporting() : PowerReporting
- GetCapSelectCharacterSet() : bool
- GetCapStatisticsReporting() : bool
- GetCapTone() : bool
- GetCapTouch() : bool
- GetCapTransaction() : bool
- GetCapUpdateFirmware() : bool
- GetCapUpdateStatistics() : bool
- GetCharacterSet() : int
- GetCharacterSetList() : CharacterSetList
- GetCheckHealthText() : string

Remote Order Display

The diagram shows a UML class named "RemoteOrderDisplay Interface". It has a compartment for "Methods" containing 30 listed methods, each preceded by a small icon. The methods are:

- GetClaimed() : bool
- GetClocks() : int
- GetCurrentUnitID() : int
- GetDataCount() : int
- GetDataEventEnabled() : bool
- GetDeviceControlDescription() : string
- GetDeviceControlVersion() : UposVersion
- GetDeviceEnabled() : bool
- GetDeviceServiceDescription() : string
- GetDeviceServiceVersion() : UposVersion
- GetErrorString() : string
- GetErrorUnits() : int
- GetEventString() : string
- GetEventType() : int
- GetEventUnitID() : int
- GetEventUnits() : int
- GetFreezeEvents() : bool
- GetMapCharacterSet() : bool
- GetOutputID() : int
- GetPhysicalDeviceDescription() : string
- GetPhysicalDeviceName() : string
- GetPowerNotify() : PowerNotification
- GetPowerState() : PowerState
- GetState() : ControlState
- GetSystemClocks() : int
- GetSystemVideoSaveBuffers() : int

Remote Order Display

The diagram shows a UML class named "RemoteOrderDisplay Interface". It has a compartment for "Methods" containing 20 listed methods. Each method is preceded by a small icon of a person with a video camera.

- Methods
 - GetTimeout() : int
 - GetUnitsOnline() : int
 - GetVideoDataCount() : int
 - GetVideoMode() : int
 - GetVideoModesList() : VideoModesList
 - GetVideoSaveBuffers() : int
 - Open(string EndpointAddress) : void
 - Release() : void
 - ResetStatistics(StatisticList StatisticsBuffer) : void
 - ResetVideo(int Units) : void
 - RestoreVideoRegion(int Units, int TargetRow, int TargetColumn, int BufferId) : void
 - RetrieveStatistics(StatisticList StatisticsBuffer) : string
 - SaveVideoRegion(int Units, int Row, int Column, int Height, int Width, int BufferId) : void
 - SelectCharacterSet(int Units, int CharacterSet) : void
 - SetAsyncMode(bool AsyncMode) : void
 - SetAutoToneDuration(int AutoToneDuration) : void
 - SetAutoToneFrequency(int AutoToneFrequency) : void
 - SetCurrentUnitID(int CurrentUnitID) : void
 - SetCursor(int Units, int Row, int Column) : void
 - SetDataEventEnabled(bool DataEventEnabled) : void

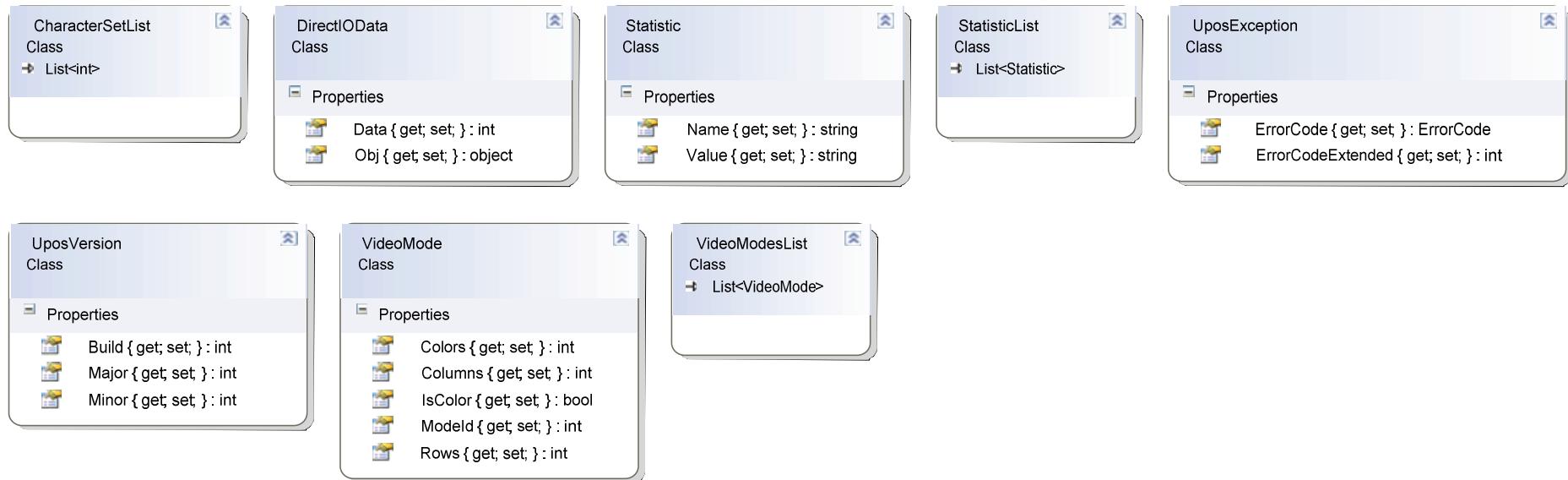
Remote Order Display

RemoteOrderDisplay
Interface

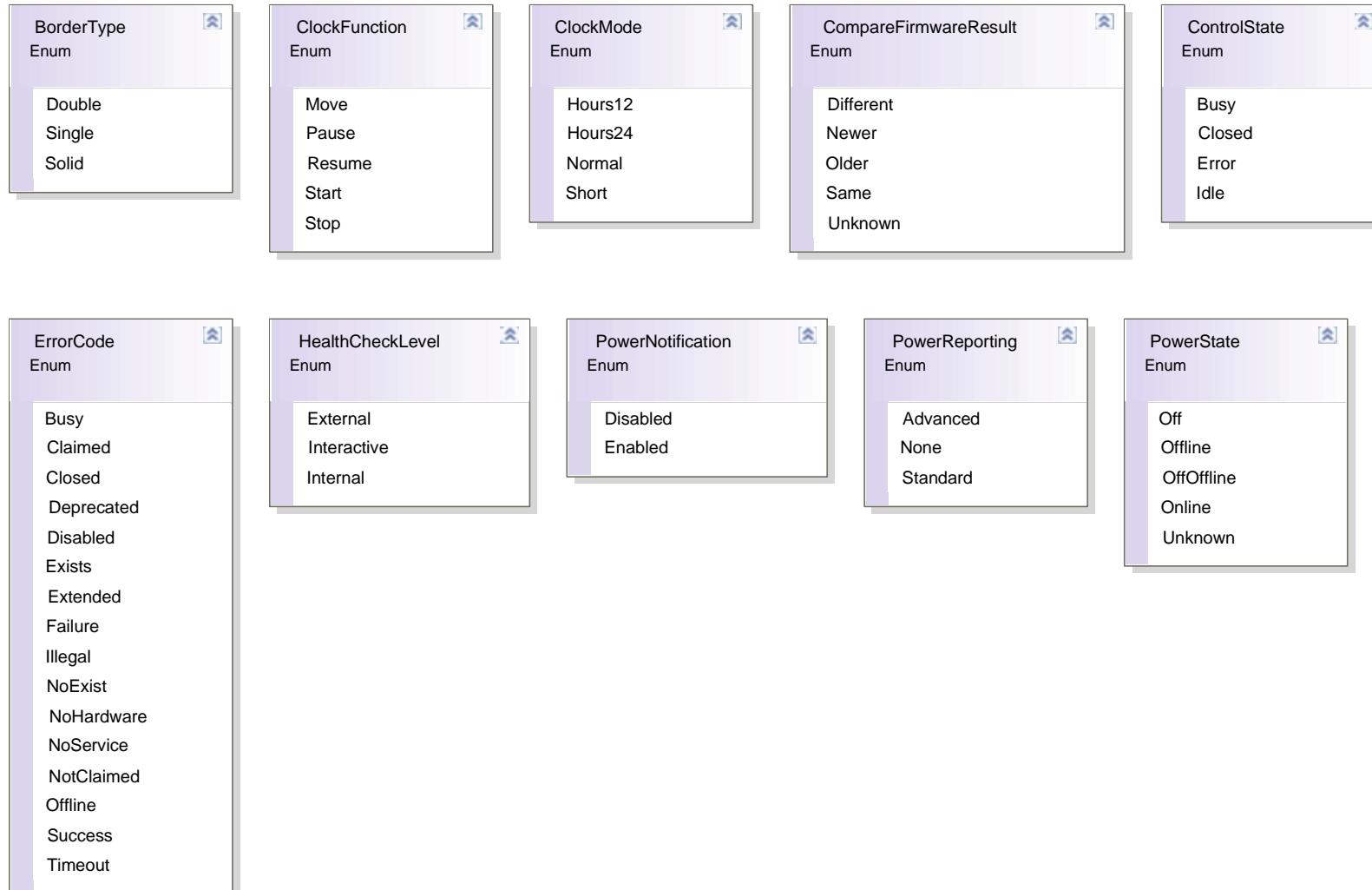
Methods

- SetDeviceEnabled(bool DeviceEnabled) : void
- SetEventType(int EventType) : void
- SetFreezeEvents(bool FreezeEvents) : void
- SetMapCharacterSet(bool MapCharacterSet) : void
- SetPowerNotify(PowerNotification PowerNotify) : void
- SetTimeout(int Timeout) : void
- SetVideoMode(int VideoMode) : void
- TransactionDisplay(int Units, RemoteOrderDisplayTransaction TransactionFunction) : void
- UpdateFirmware(string FirmwareFileName) : void
- UpdateStatistics(StatisticList StatisticsBuffer) : void
- UpdateVideoRegionAttribute(int Units, VideoAttributeCommand AttributeFunction, int Column, int Height, int Width, int Attribu te) : void
- VideoSound(int Units, int Frequency, int Duration, int NumberOfCycles, int InterSoundWait) : void

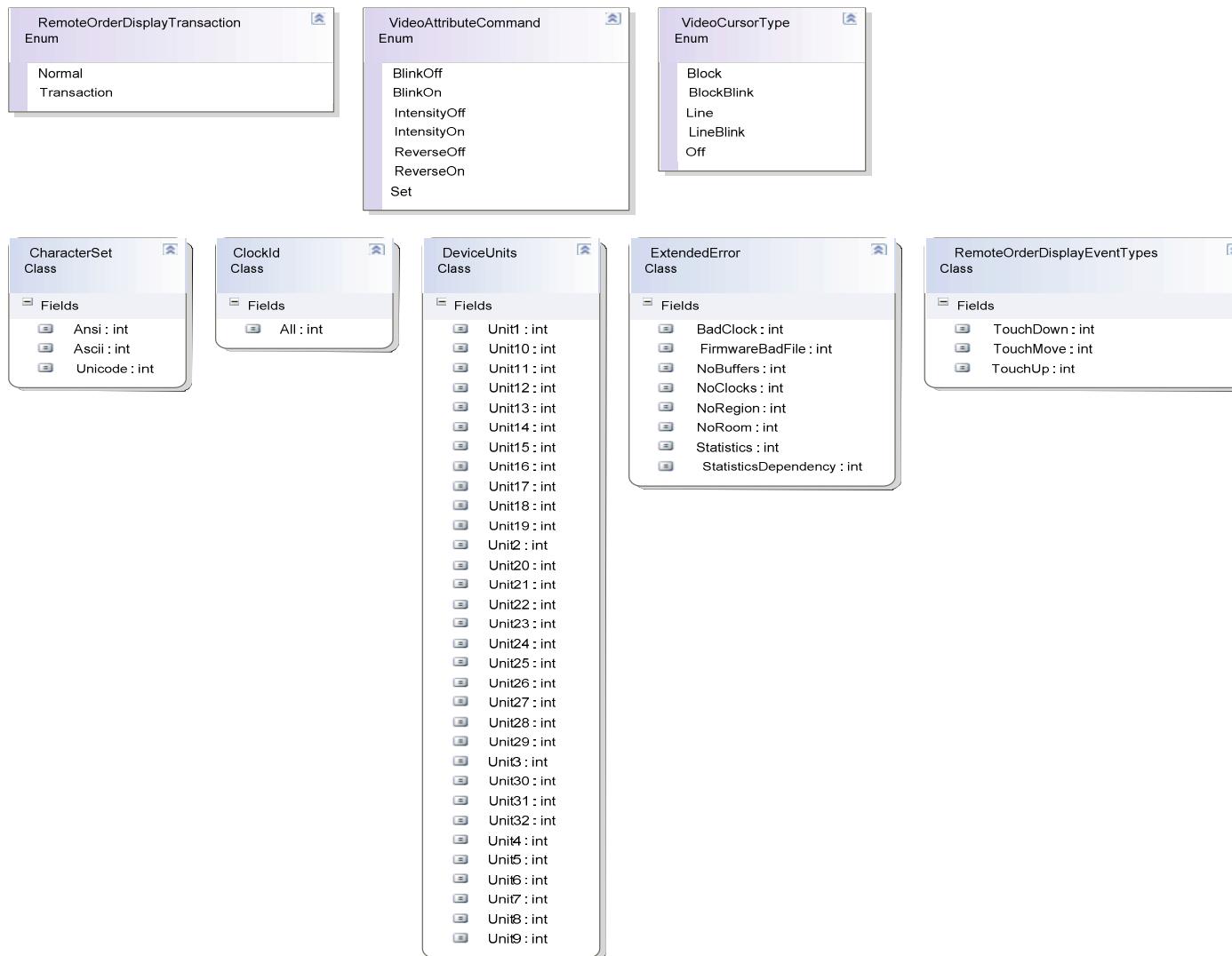
Remote Order Display



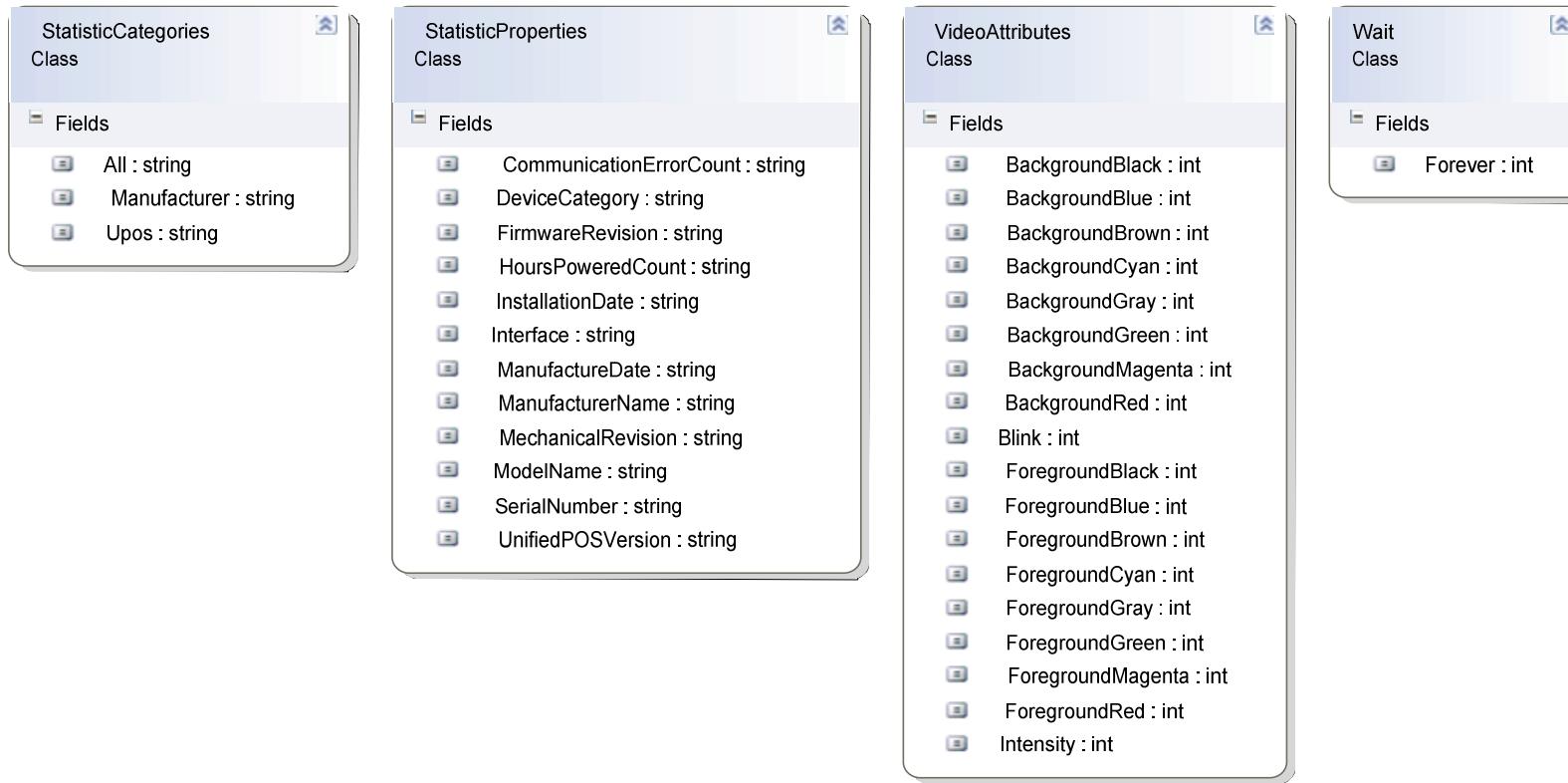
Remote Order Display



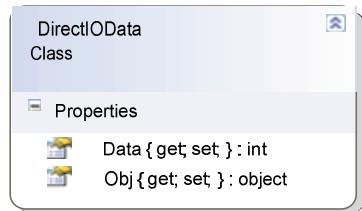
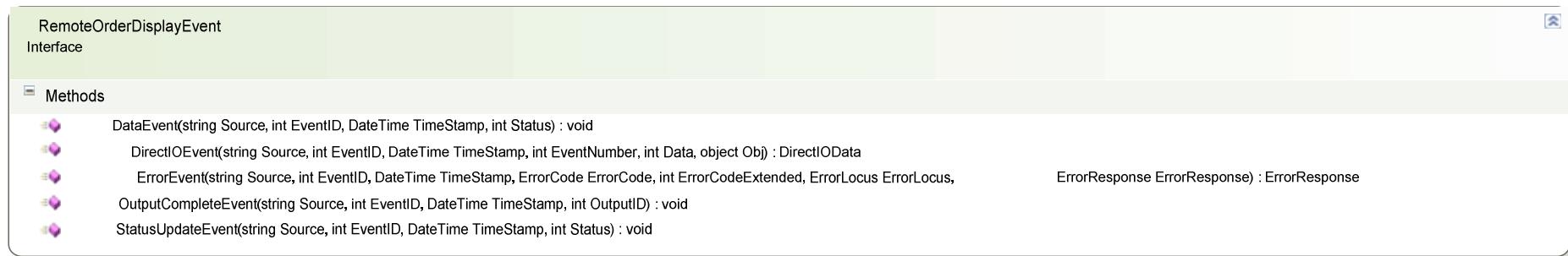
Remote Order Display



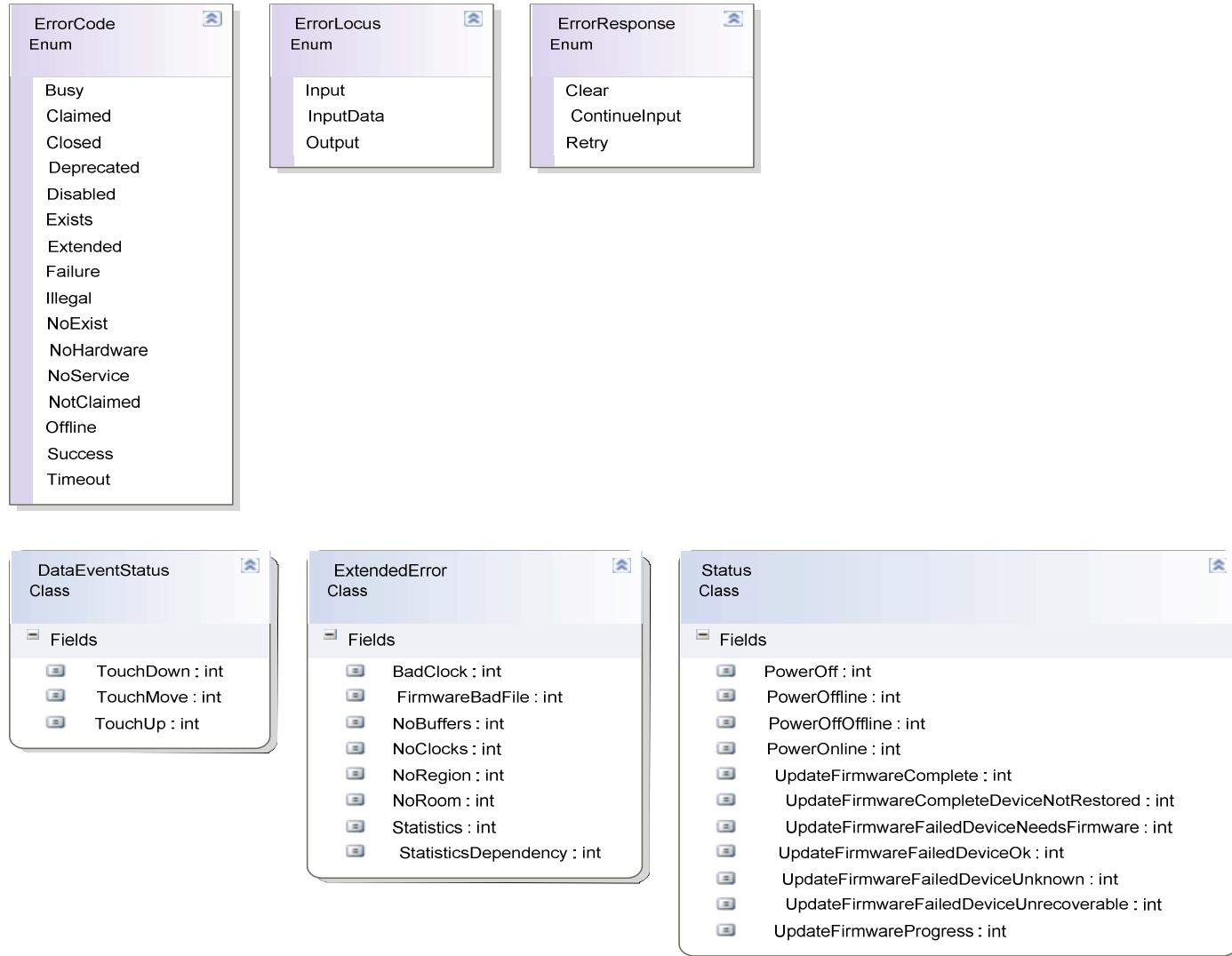
Remote Order Display



Remote Order Display Events



Remote Order Display Events



Radio Frequency Identification Scanner (RFID Scanner)

The diagram shows a UML class named 'RFIDScanner' with a green background, indicating it is an interface. It has a single method listed under the 'Methods' section.

RFIDScanner
Interface

Methods

- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- ClearInput() : void
- ClearInputProperties() : void
- ClearOutput() : void
- Close(string EndpointAddress) : void
 - CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
- DirectIO(int Command, int Data, object Obj) : DirectIOData
- DisableTag(byte[] TagID, int Timeout, byte[] Password) : void
- FirstTag() : void
- GetAutoDisable() : bool
- GetCapCompareFirmwareVersion() : bool
- GetCapContinuousRead() : bool
- GetCapDisableTag() : bool
- GetCapLockTag() : bool
- GetCapMultipleProtocols() : int
- GetCapPowerReporting() : PowerReporting
- GetCapReadTimer() : bool
- GetCapStatisticsReporting() : bool
- GetCapUpdateFirmware() : bool
- GetCapUpdateStatistics() : bool
- GetCapWriteTag() : WriteTagSections
- GetCheckHealthText() : string
- GetClaimed() : bool
- GetContinuousReadMode() : bool
- GetCurrentTagID() : byte[]
- GetCurrentTagProtocol() : int

Radio Frequency Identification Scanner (RFID Scanner)

RFIDScanner
Interface



Methods

- GetCurrentTagUserData() : byte[]
- GetDataCount() : int
- GetDataEventEnabled() : bool
- GetDeviceControlDescription() : string
- GetDeviceControlVersion() : UposVersion
- GetDeviceEnabled() : bool
- GetDeviceServiceDescription() : string
- GetDeviceServiceVersion() : UposVersion
- GetFreezeEvents() : bool
- GetOutputID() : int
- GetPhysicalDeviceDescription() : string
- GetPhysicalDeviceName() : string
- GetPowerNotify() : PowerNotification
- GetPowerState() : PowerState
- GetProtocolMask() : int
- GetReadTimerInterval() : int
- GetState() : ControlState
- GetTagCount() : int
- LockTag(byte[] TagID, int Timeout, byte[] Password) : void
- NextTag() : void
- Open(string EndpointAddress) : void
- PreviousTag() : void
- ReadTags(RFIDReadOptions Cmd, byte[] FilterID, byte[] FilterMask, int Start, int Length, int Timeout, byte[] Password) : void

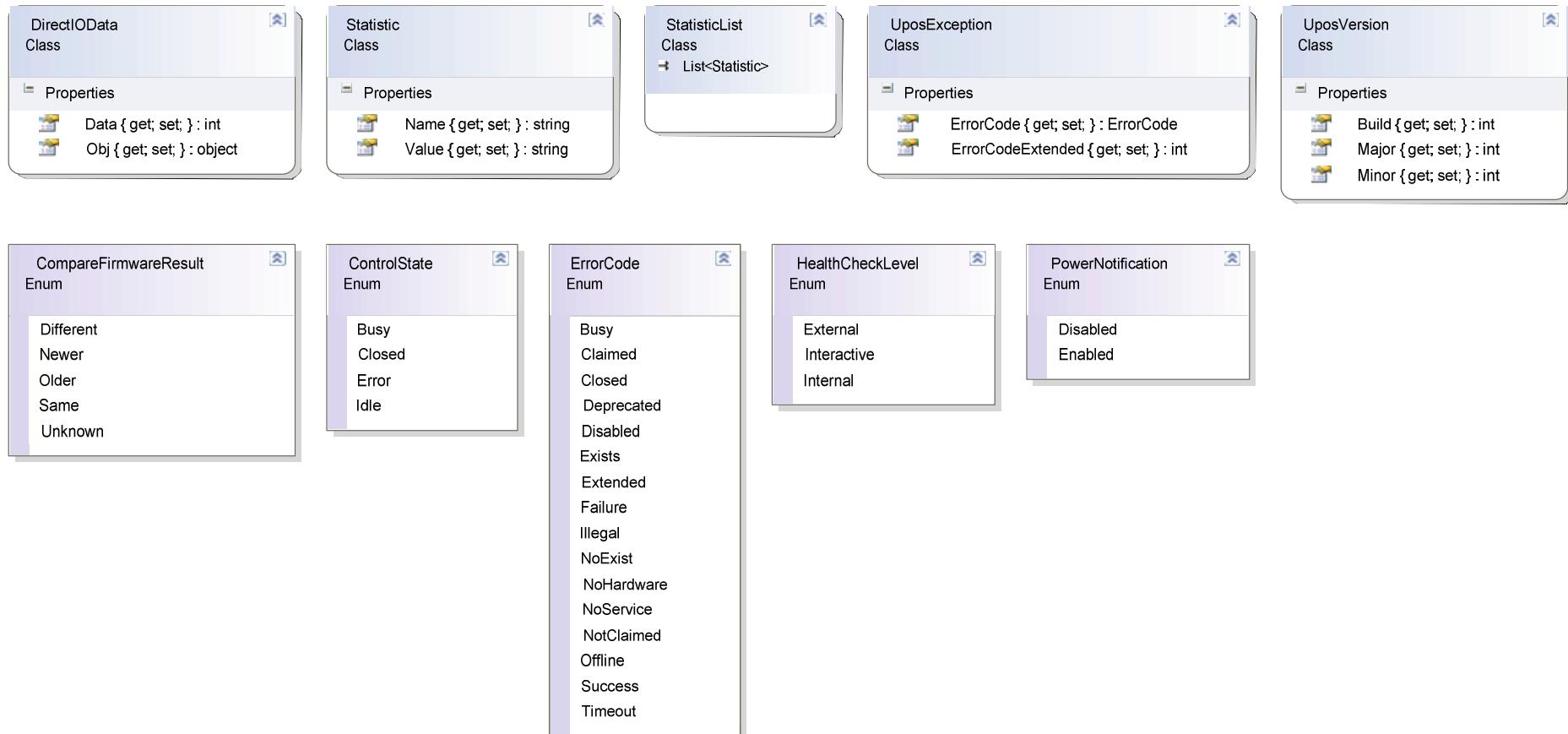
Radio Frequency Identification Scanner (RFID Scanner)

RFIDScanner
Interface

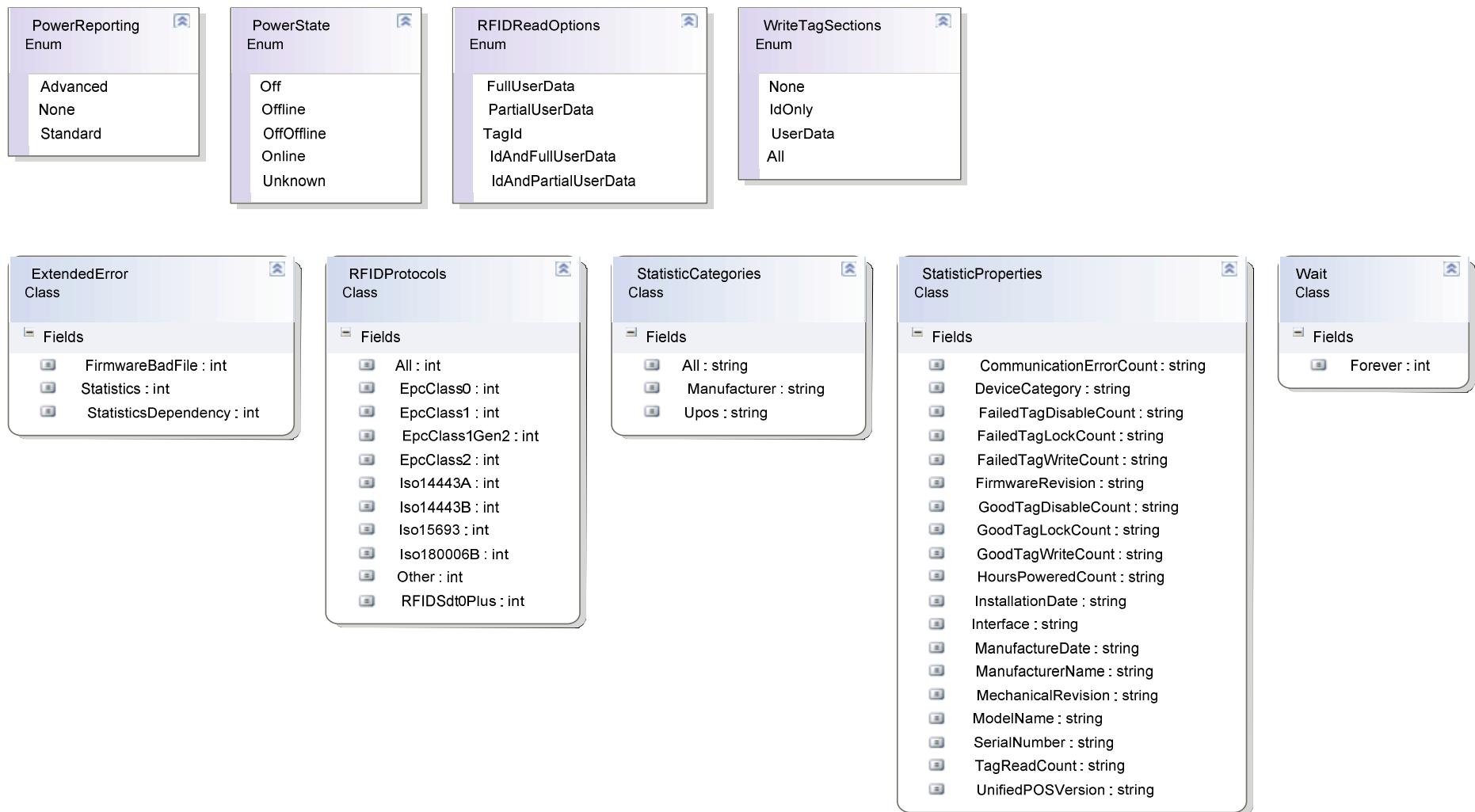
Methods

- Release() : void
- ResetStatistics(StatisticList StatisticsBuffer) : void
- RetrieveStatistics(StatisticList StatisticsBuffer) : string
- SetAutoDisable(bool AutoDisable) : void
- SetDataEventEnabled(bool DataEventEnabled) : void
- SetDeviceEnabled(bool DeviceEnabled) : void
- SetFreezeEvents(bool FreezeEvents) : void
- SetPowerNotify(PowerNotification PowerNotify) : void
- SetProtocolMask(int ProtocolMask) : void
- SetReadTimerInterval(int ReadTimerInterval) : void
 - StartReadTags(RFIDReadOptions Cmd, byte[] FilterID, byte[] FilterMask, int Start, int Length, byte[] Password) : void
- StopReadTags(byte[] Password) : void
- UpdateFirmware(string FirmwareFileName) : void
- UpdateStatistics(StatisticList StatisticsBuffer) : void
 - WriteTagData(byte[] TagID, byte[] UserData, int Start, int Timeout, byte[] Password) : void
 - WriteTagID(byte[] SourceTagID, byte[] DestinationTagID, int Timeout, byte[] Password) : void

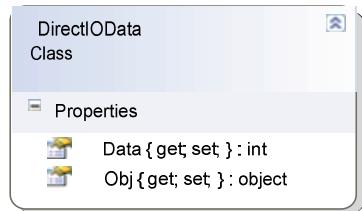
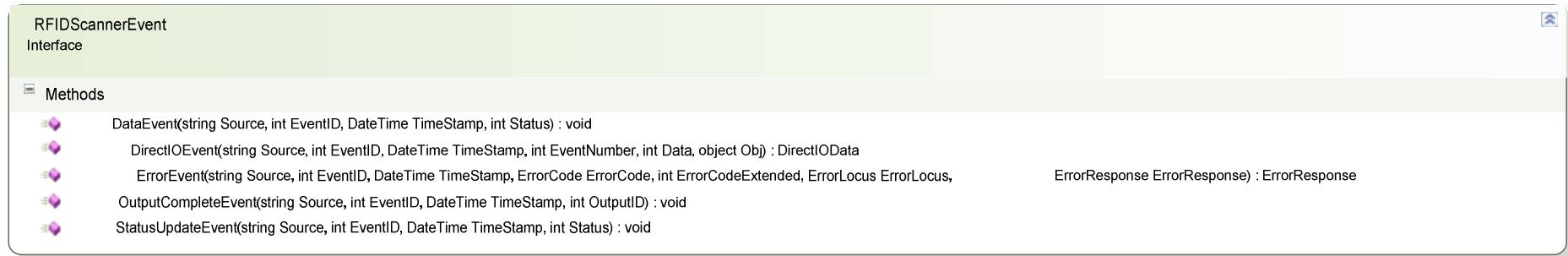
Radio Frequency Identification Scanner (RFID Scanner)



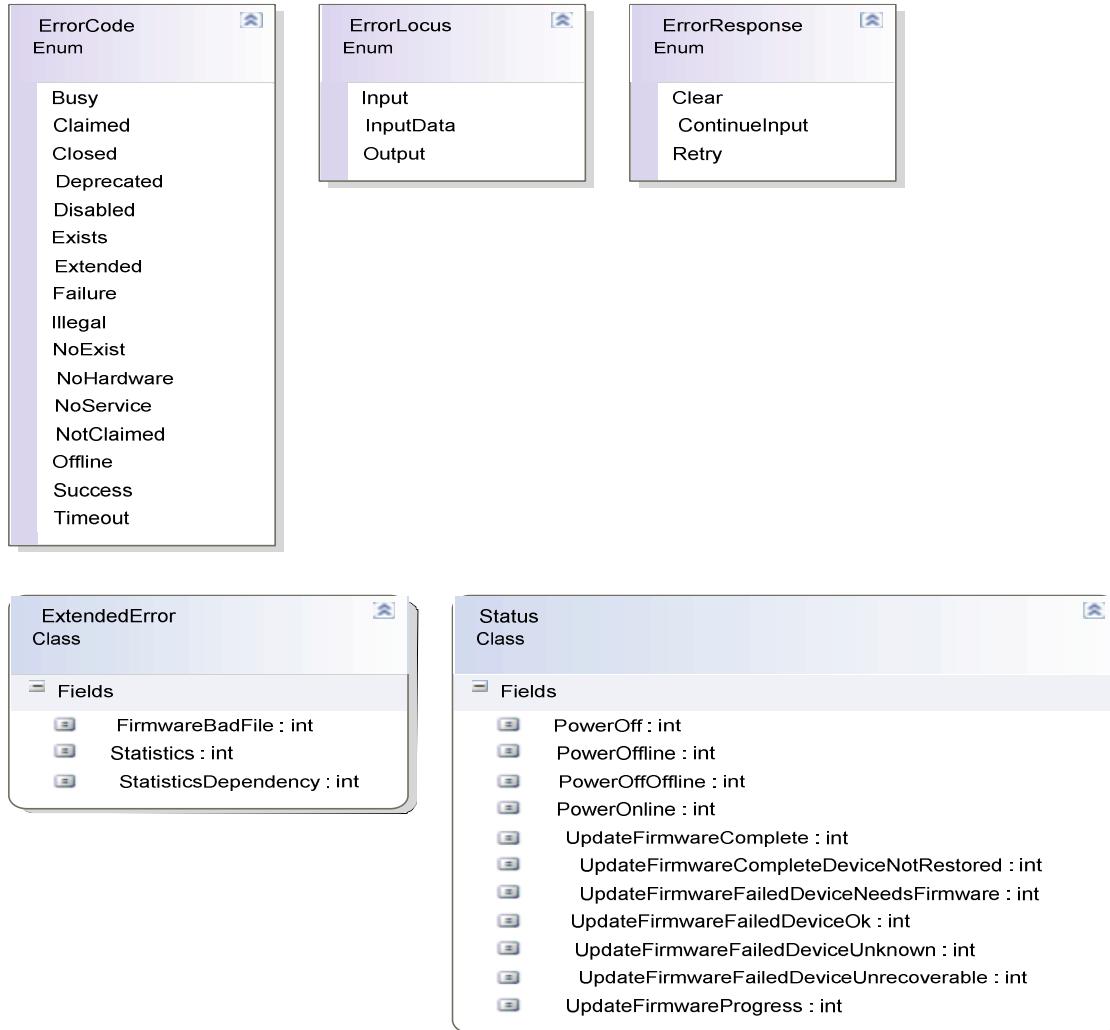
Radio Frequency Identification Scanner (RFID Scanner)



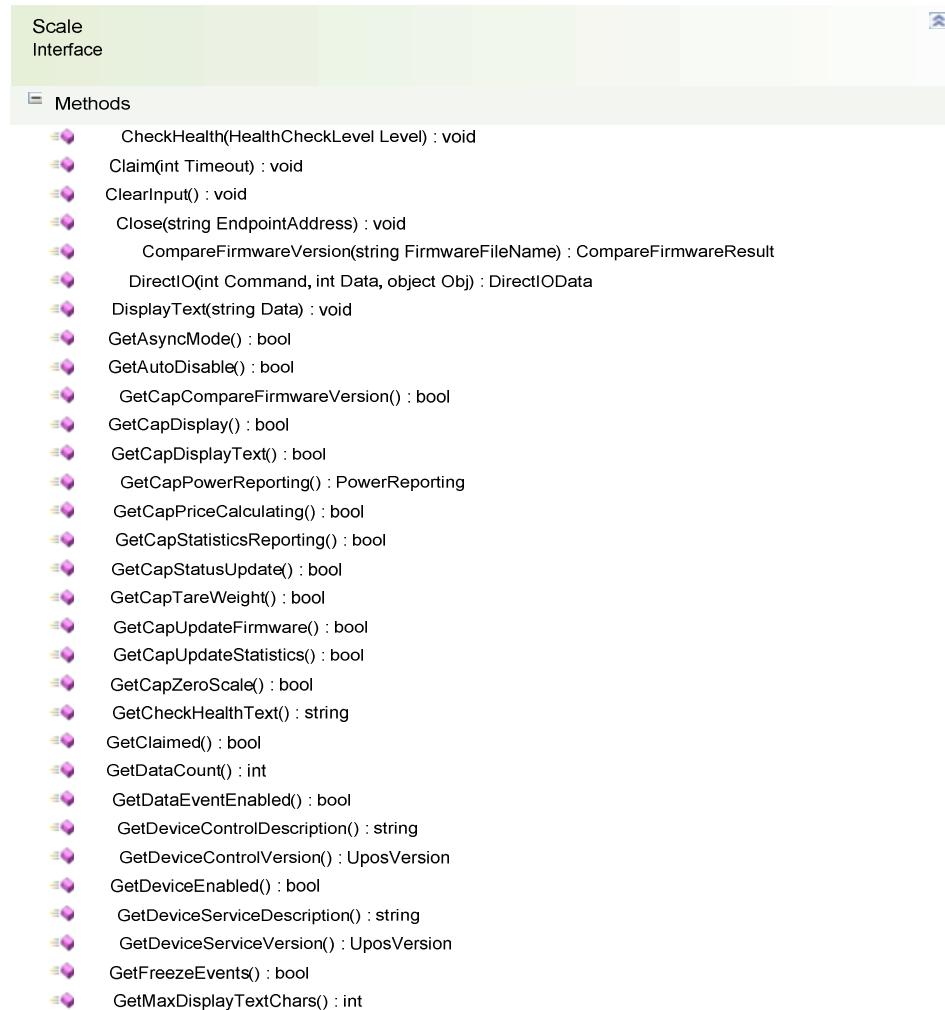
Radio Frequency Identification Scanner (RFID Scanner) Events



Radio Frequency Identification Scanner (RFID Scanner) Events



Scale

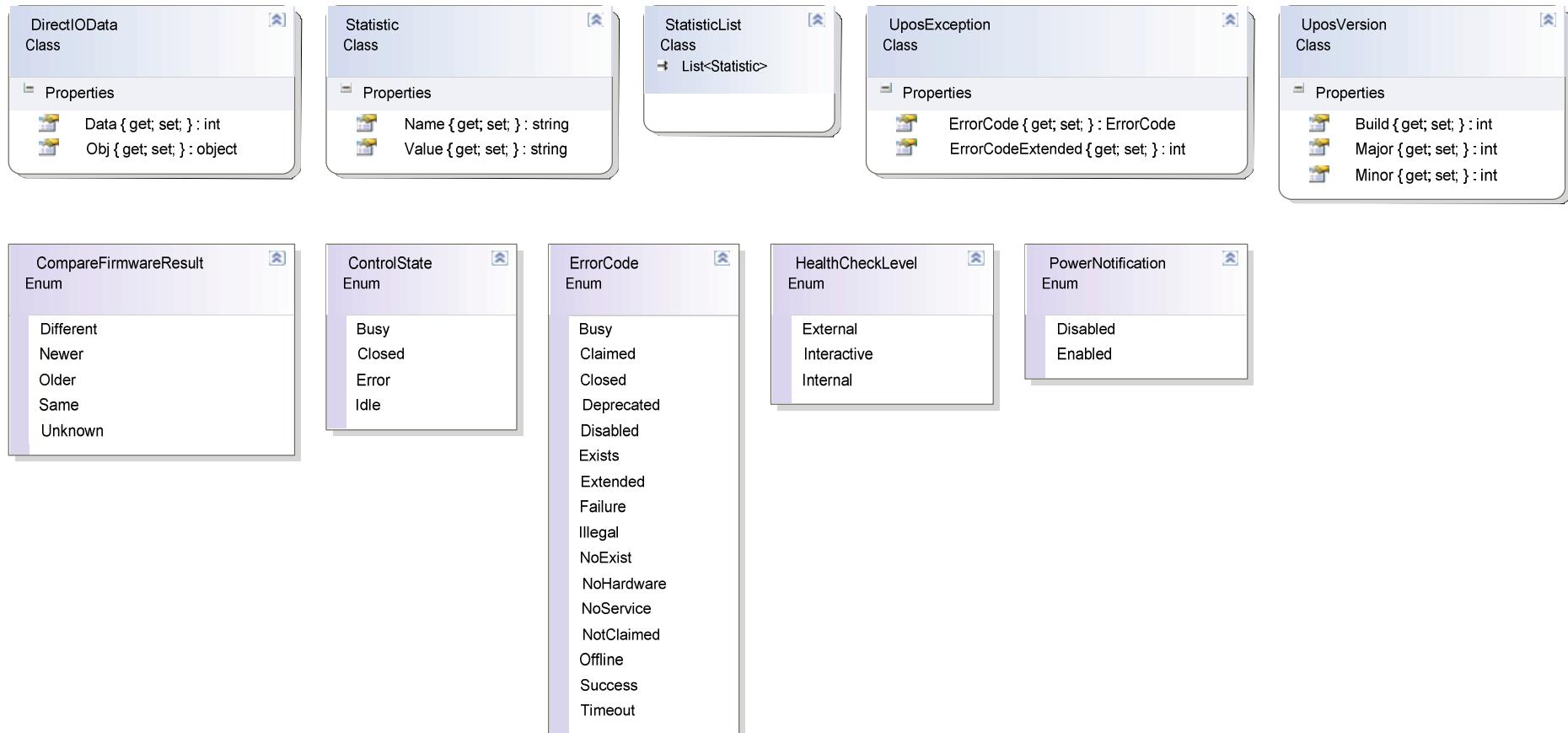


Scale



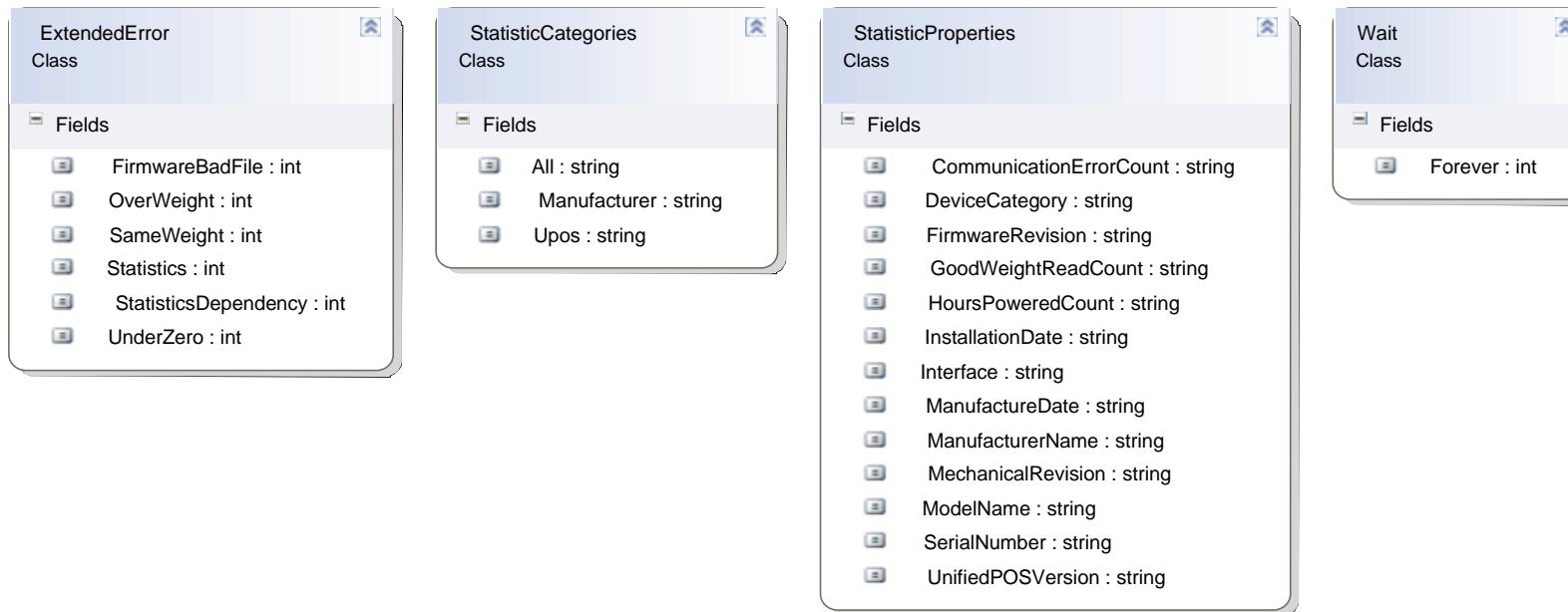
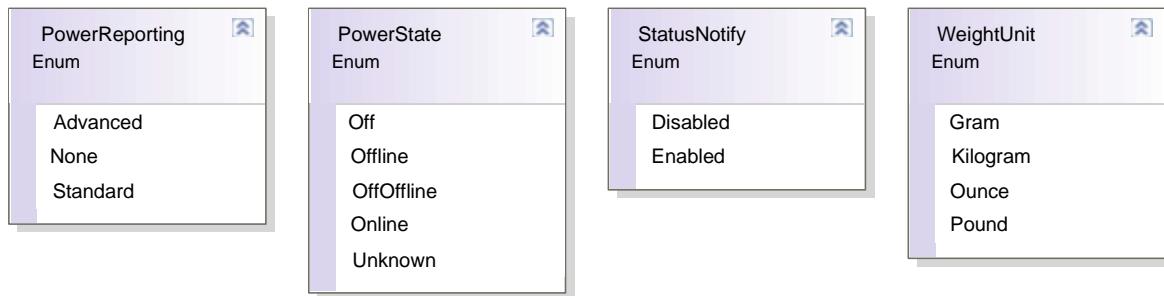
- » GetMaximumWeight() : decimal
- » GetPhysicalDeviceDescription() : string
- » GetPhysicalDeviceName() : string
- » GetPowerNotify() : PowerNotification
- » GetPowerState() : PowerState
- » GetSalesPrice() : decimal
- » GetScaleLiveWeight() : decimal
- » GetState() : ControlState
- » GetStatusNotify() : StatusNotify
- » GetTareWeight() : decimal
- » GetUnitPrice() : decimal
- » GetWeightUnit() : WeightUnit
- » GetZeroValid() : bool
- » Open(string EndpointAddress) : void
- » ReadWeight(int Timeout) : decimal
- » Release() : void
- » ResetStatistics(StatisticList StatisticsBuffer) : void
- » RetrieveStatistics(StatisticList StatisticsBuffer) : string
- » SetAsyncMode(bool AsyncMode) : void
- » SetAutoDisable(bool AutoDisable) : void
- » SetDataEventEnabled(bool DataEventEnabled) : void
- » SetDeviceEnabled(bool DeviceEnabled) : void
- » SetFreezeEvents(bool FreezeEvents) : void
- » SetPowerNotify(PowerNotification PowerNotify) : void
- » SetStatusNotify(StatusNotify StatusNotify) : void
- » SetTareWeight(decimal TareWeight) : void
- » SetUnitPrice(decimal UnitPrice) : void
- » SetZeroValid(bool ZeroValid) : void
- » UpdateFirmware(string FirmwareFileName) : void
- » UpdateStatistics(StatisticList StatisticsBuffer) : void
- » ZeroScale() : void

Scale



WS-POS 1.1 Technical Specification

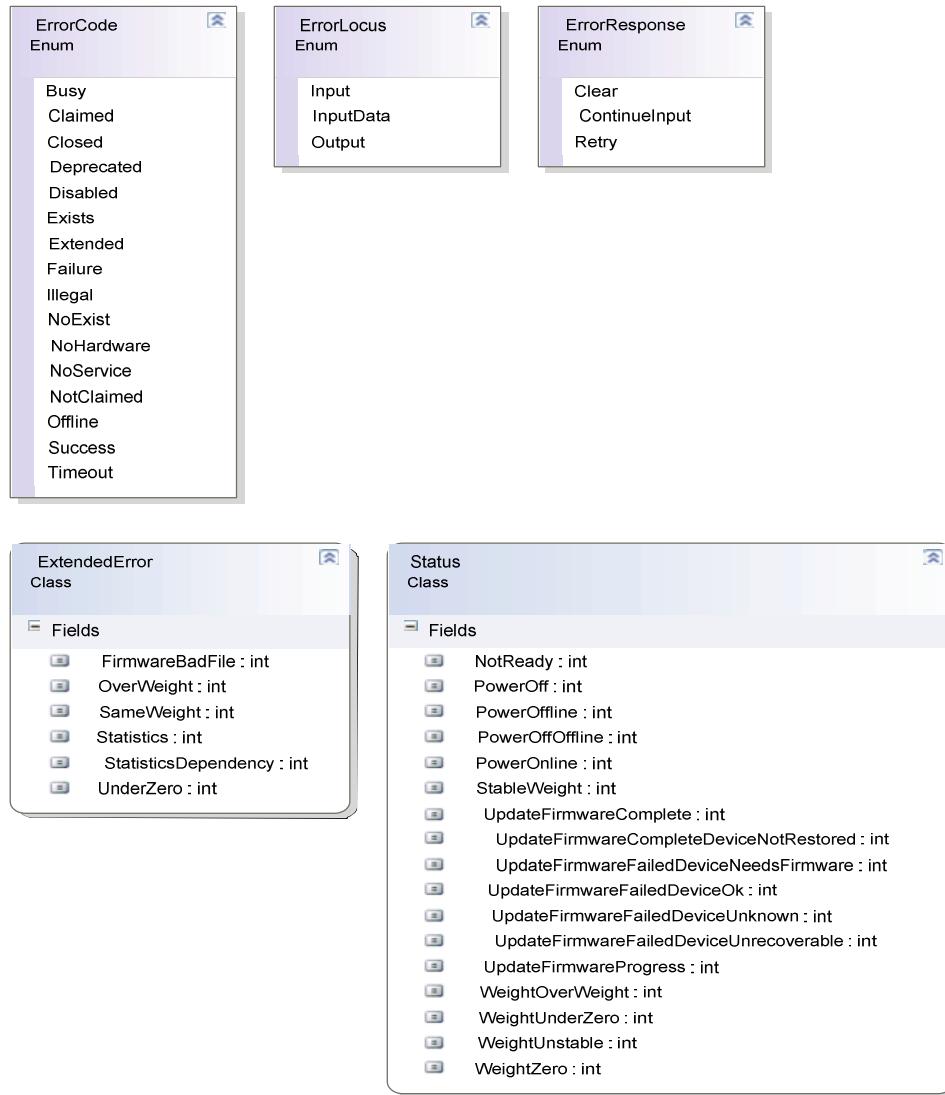
Scale



Scale Events



Scale Events



Scanner (Bar Code)

The diagram shows a UML class named "Scanner" with a green background. It has a section titled "Methods" containing a list of 34 methods, each preceded by a small purple icon. The methods are:

- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- ClearInput() : void
- ClearInputProperties() : void
- Close(string EndpointAddress) : void
- CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
- DirectIO(int Command, int Data, object Obj) : DirectIOData
- GetAutoDisable() : bool
- GetCapCompareFirmwareVersion() : bool
- GetCapPowerReporting() : PowerReporting
- GetCapStatisticsReporting() : bool
- GetCapUpdateFirmware() : bool
- GetCapUpdateStatistics() : bool
- GetCheckHealthText() : string
- GetClaimed() : bool
- GetDataCount() : int
- GetDataEventEnabled() : bool
- GetDecodeData() : bool
- GetDeviceControlDescription() : string
- GetDeviceControlVersion() : UposVersion
- GetDeviceEnabled() : bool
- GetDeviceServiceDescription() : string
- GetDeviceServiceVersion() : UposVersion
- GetFreezeEvents() : bool
- GetPhysicalDeviceDescription() : string

Scanner (Bar Code)

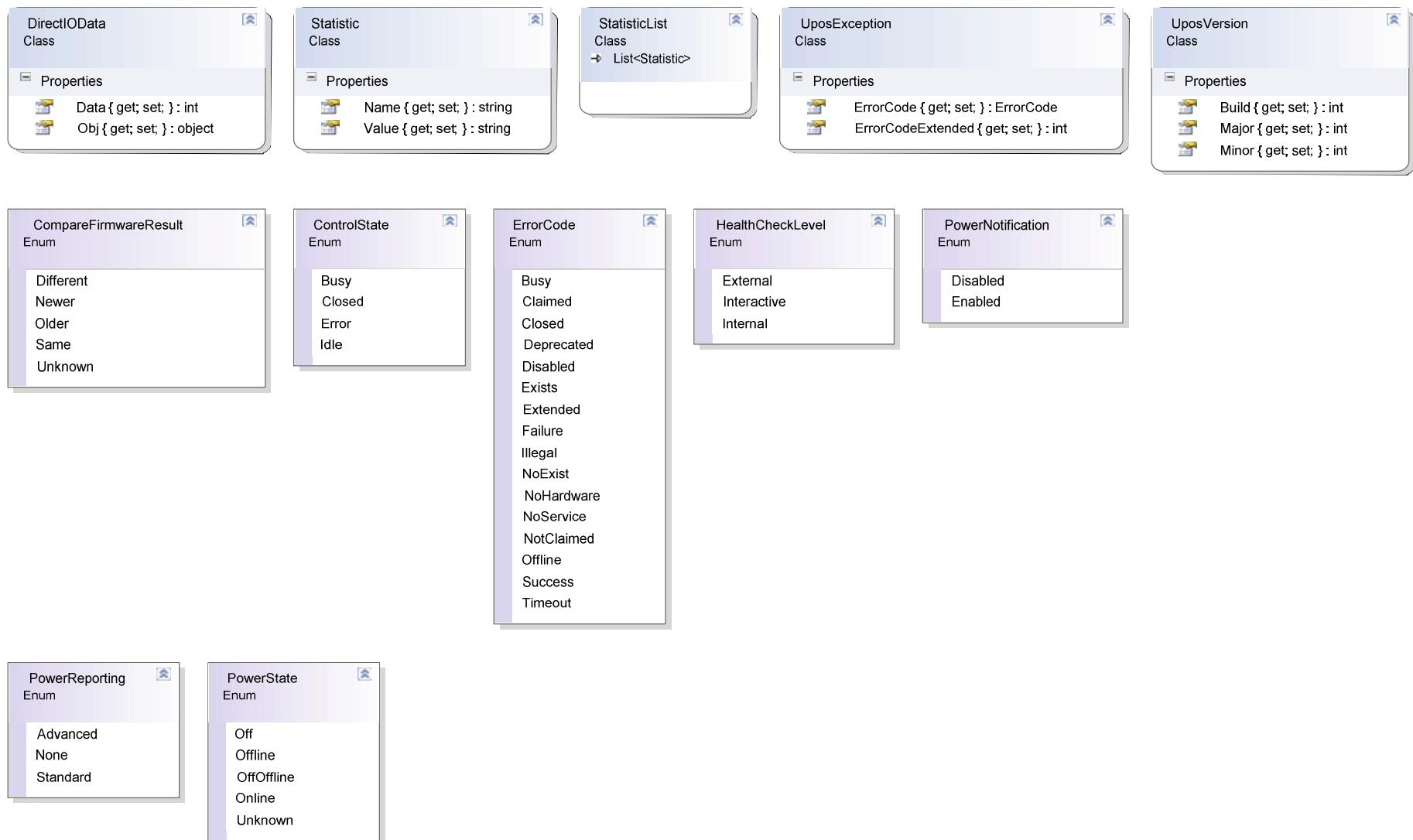


Methods

- GetPhysicalDeviceName() : string
- GetPowerNotify() : PowerNotification
- GetPowerState() : PowerState
- GetScanData() : byte[]
- GetScanDataLabel() : byte[]
- GetScanDataType() : int
- GetState() : ControlState
- Open(string EndpointAddress) : void
- Release() : void
- ResetStatistics(StatisticList StatisticsBuffer) : void
- RetrieveStatistics(StatisticList StatisticsBuffer) : string
- SetAutoDisable(bool AutoDisable) : void
- SetDataEventEnabled(bool DataEventEnabled) : void
- SetDecodeData(bool DecodeData) : void
- SetDeviceEnabled(bool DeviceEnabled) : void
- SetFreezeEvents(bool FreezeEvents) : void
- SetPowerNotify(PowerNotification PowerNotify) : void
- UpdateFirmware(string FirmwareFileName) : void
- UpdateStatistics(StatisticList StatisticsBuffer) : void

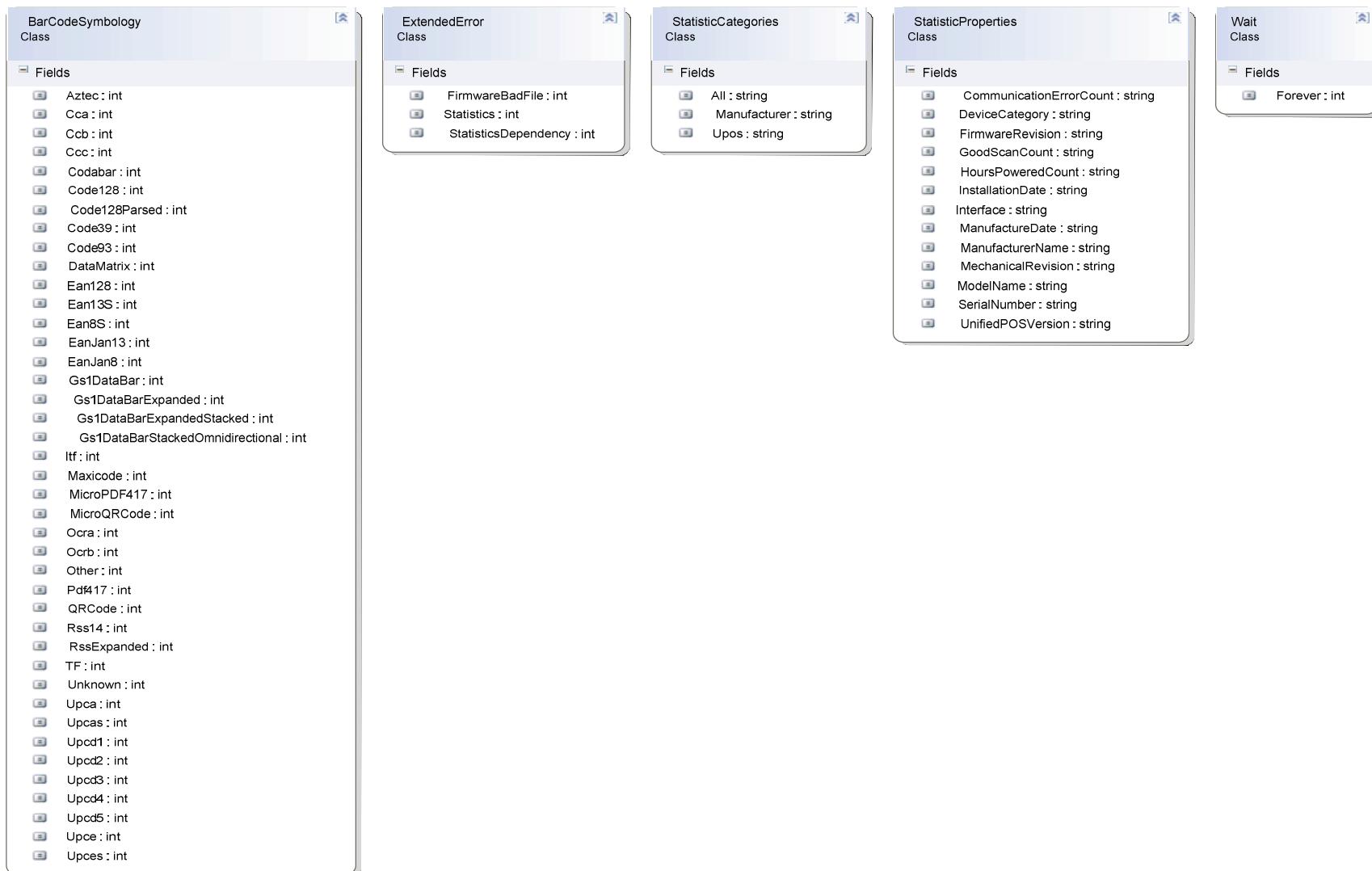
WS-POS 1.1 Technical Specification

Scanner (Bar Code)

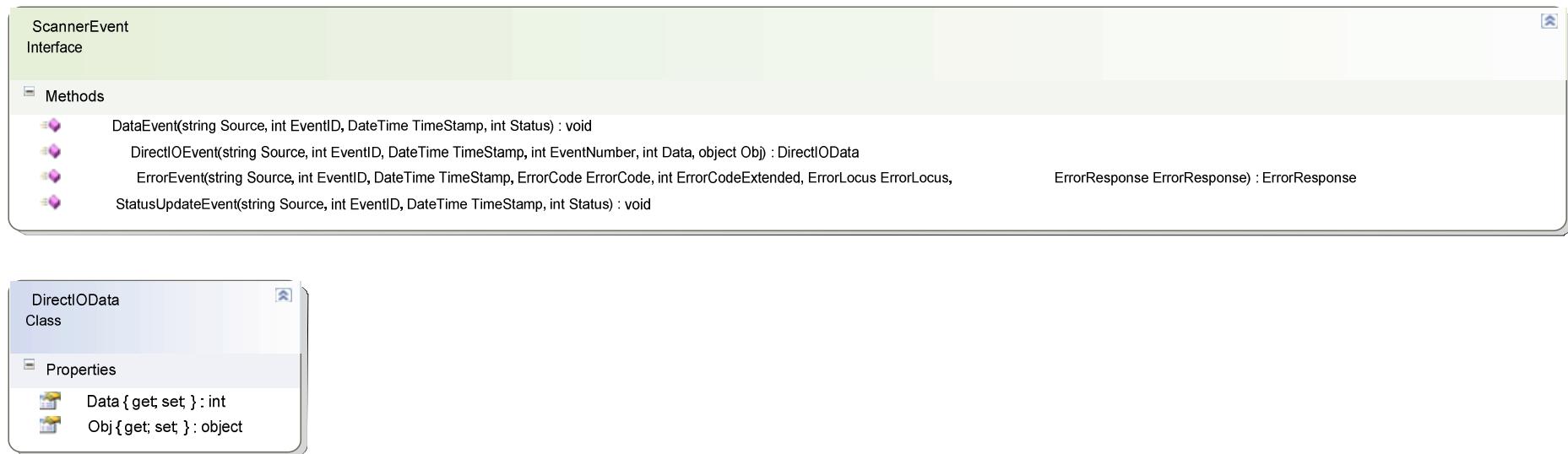


WS-POS 1.1 Technical Specification

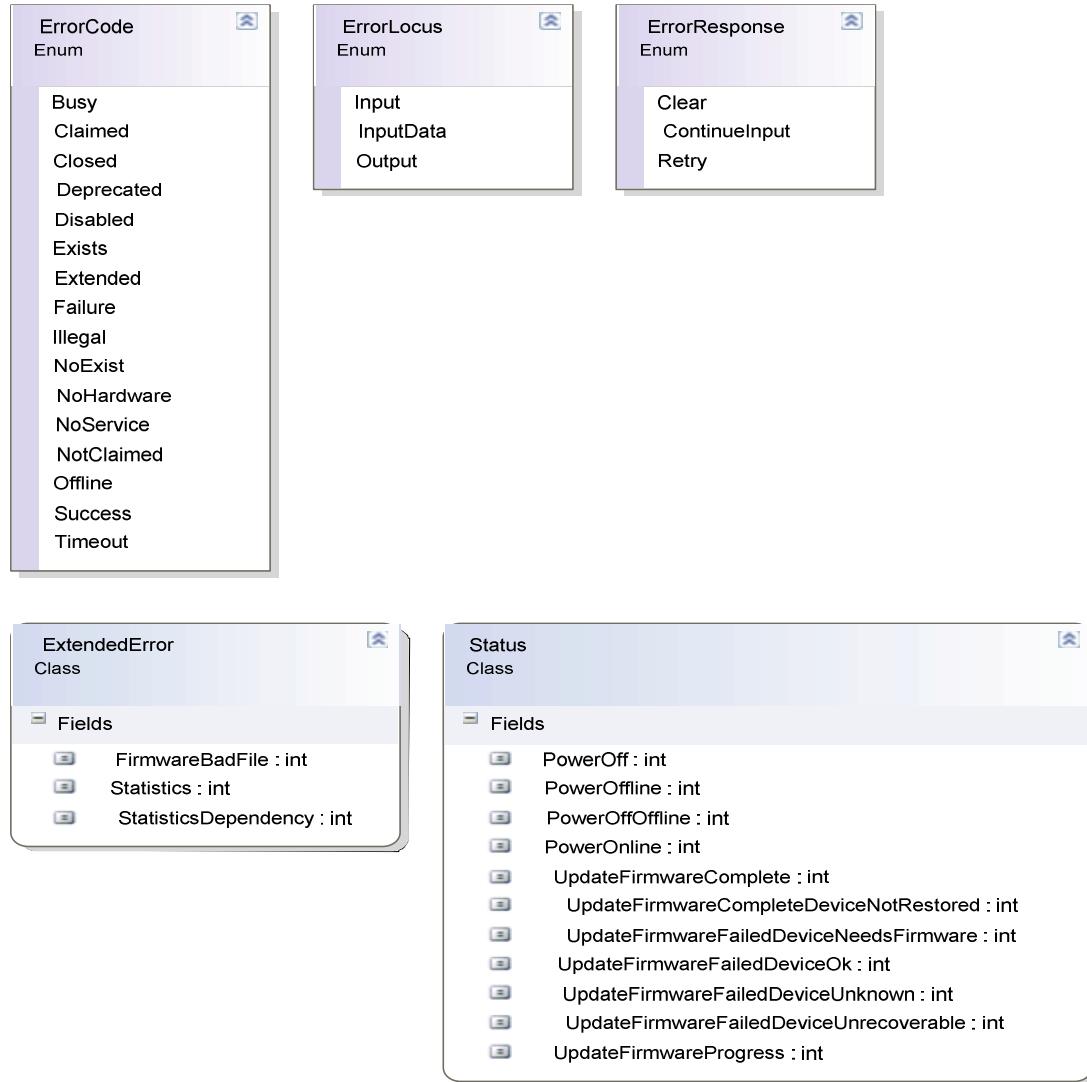
Scanner (Bar Code)



Scanner (Bar Code) Events



Scanner (Bar Code) Events

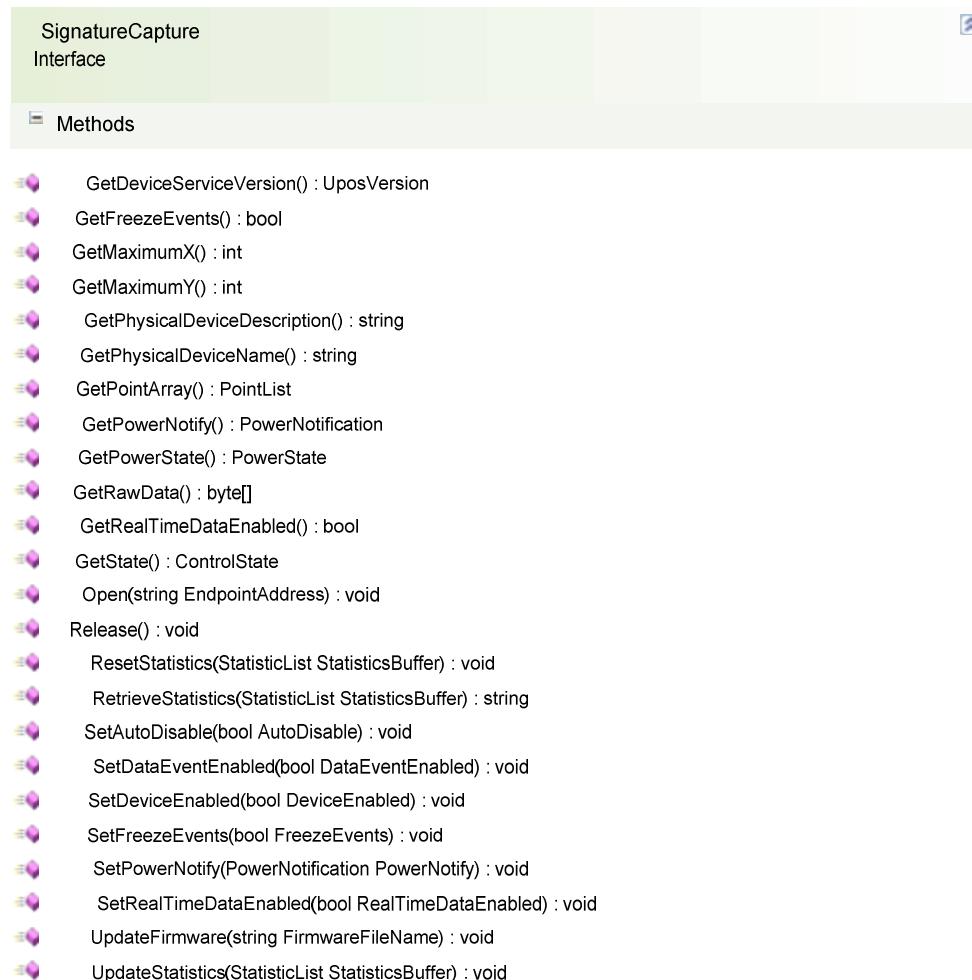


Signature Capture

The diagram shows a UML class named "SignatureCapture Interface". It has a section titled "Methods" which lists the following methods:

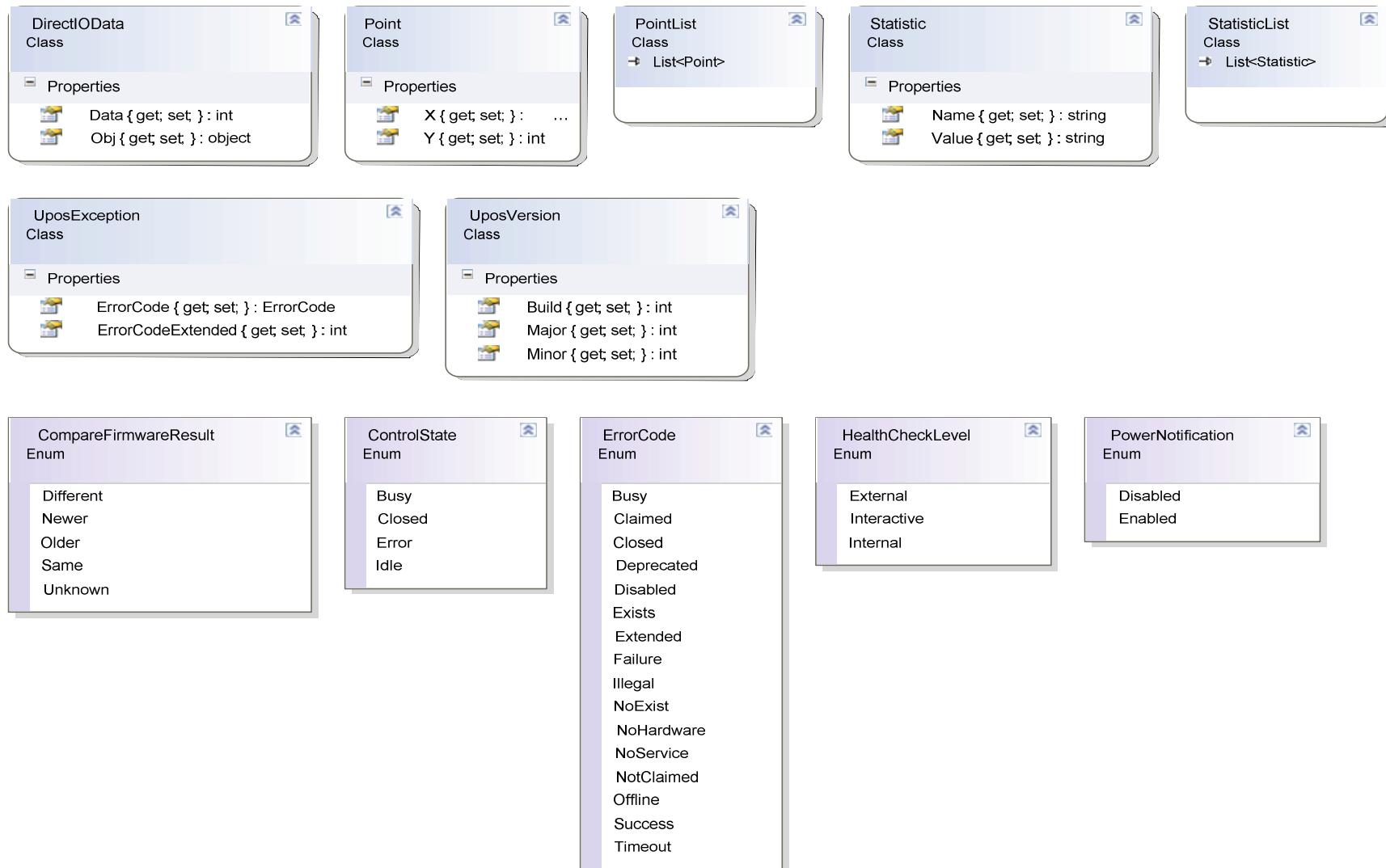
- BeginCapture(string FormName) : void
- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- ClearInput() : void
- ClearInputProperties() : void
- Close(string EndpointAddress) : void
 - CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
 - DirectIO(int Command, int Data, object Obj) : DirectIOData
- EndCapture() : void
- GetAutoDisable() : bool
- GetCapCompareFirmwareVersion() : bool
- GetCapDisplay() : bool
- GetCapPowerReporting() : PowerReporting
- GetCapRealTimeData() : bool
- GetCapStatisticsReporting() : bool
- GetCapUpdateFirmware() : bool
- GetCapUpdateStatistics() : bool
- GetCapUserTerminated() : bool
- GetCheckHealthText() : string
- GetClaimed() : bool
- GetDataCount() : int
- GetDataEventEnabled() : bool
- GetDeviceControlDescription() : string
- GetDeviceControlVersion() : UposVersion
- GetDeviceEnabled() : bool
- GetDeviceServiceDescription() : string

Signature Capture

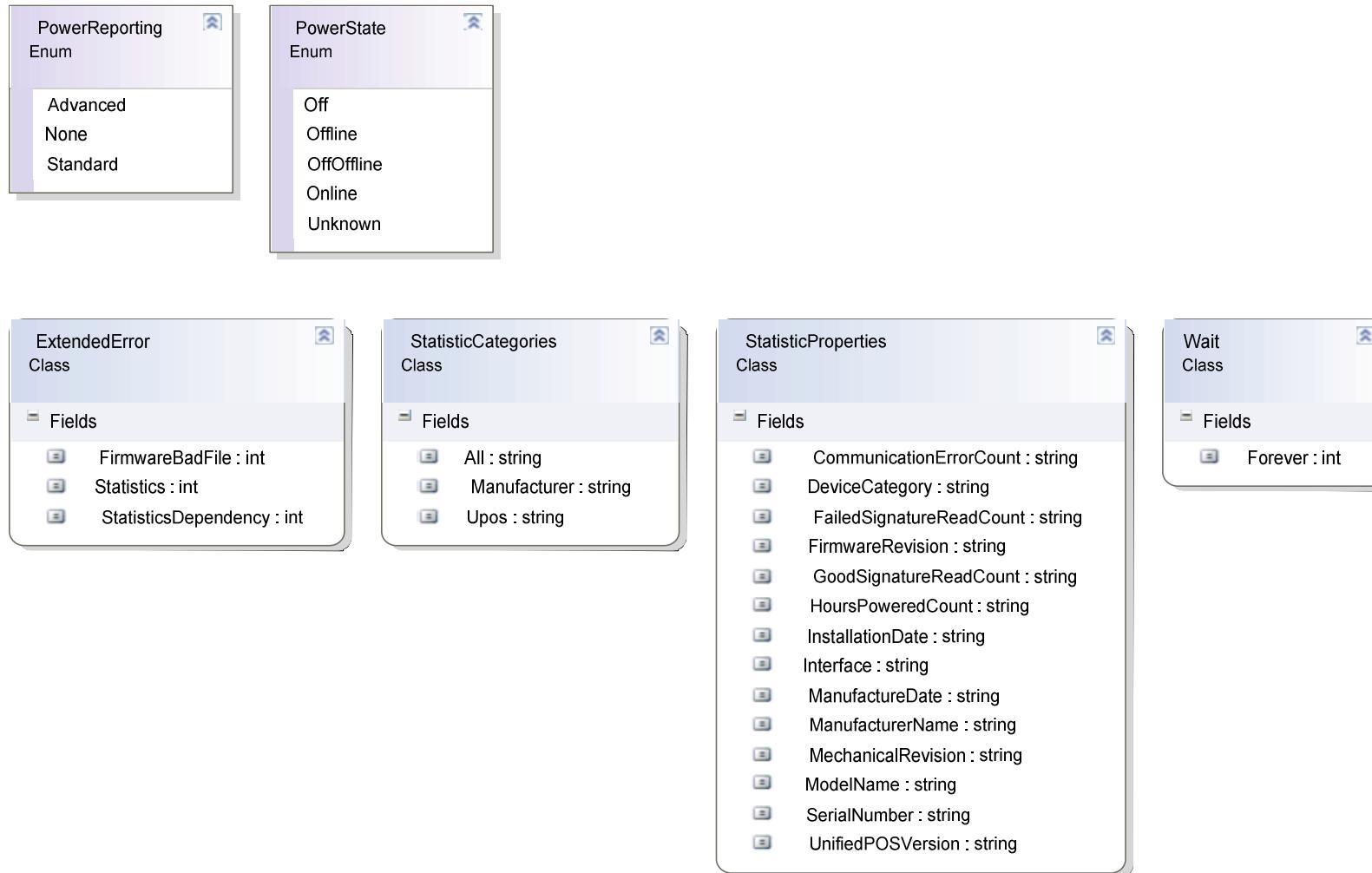


WS-POS 1.1 Technical Specification

Signature Capture



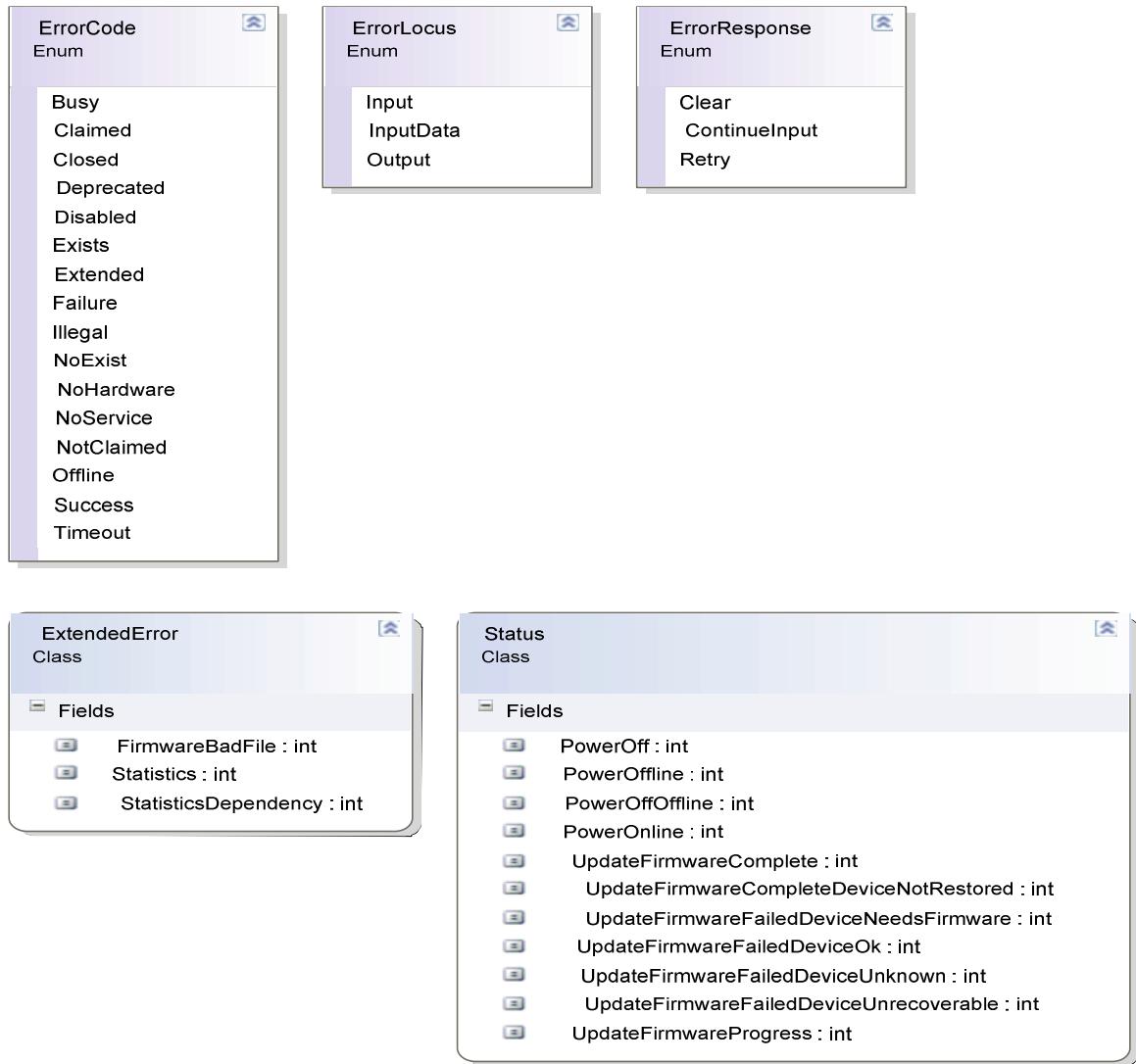
Signature Capture



Signature Capture Events



Signature Capture Events



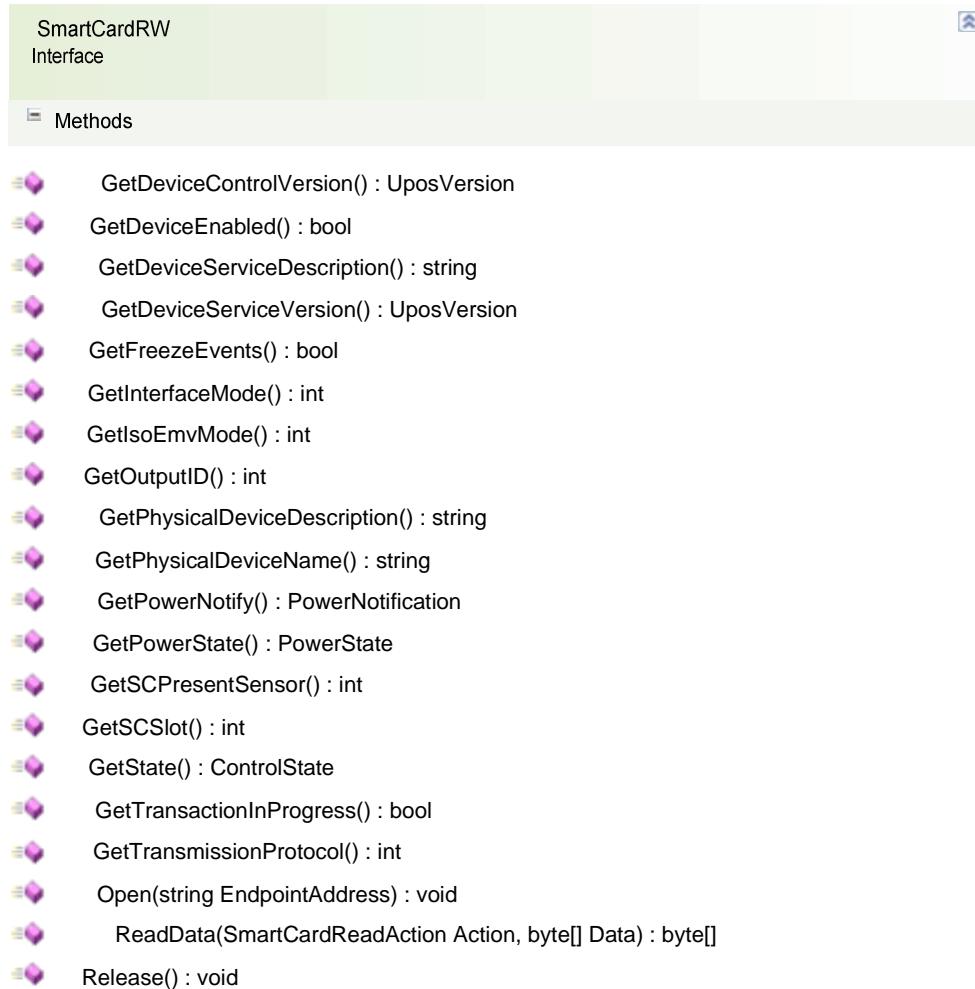
Smart Card Reader/Writer (Smart Card R/W)

SmartCardRW
Interface

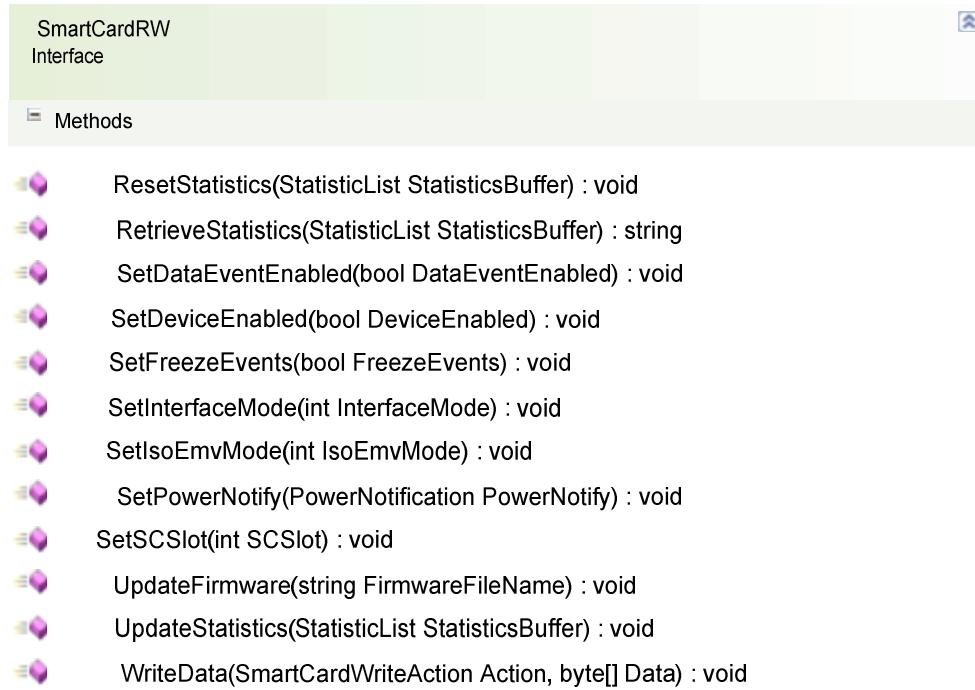
Methods

- BeginInsertion(int Timeout) : void
- BeginRemoval(int Timeout) : void
- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- ClearInput() : void
- ClearInputProperties() : void
- ClearOutput() : void
- Close(string EndpointAddress) : void
 - CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
- DirectIO(int Command, int Data, object Obj) : DirectIOData
- EndInsertion() : void
- EndRemoval() : void
 - GetCapCardErrorDetection() : bool
 - GetCapCompareFirmwareVersion() : bool
 - GetCapInterfaceMode() : int
 - GetCapIsoEmvMode() : int
 - GetCapPowerReporting() : PowerReporting
 - GetCapSCPresentSensor() : int
 - GetCapSCSlots() : int
 - GetCapStatisticsReporting() : bool
 - GetCapTransmissionProtocol() : int
 - GetCapUpdateFirmware() : bool
 - GetCapUpdateStatistics() : bool
 - GetCheckHealthText() : string
 - GetClaimed() : bool
 - GetDataCount() : int
 - GetDataEventEnabled() : bool
 - GetDeviceControlDescription() : string

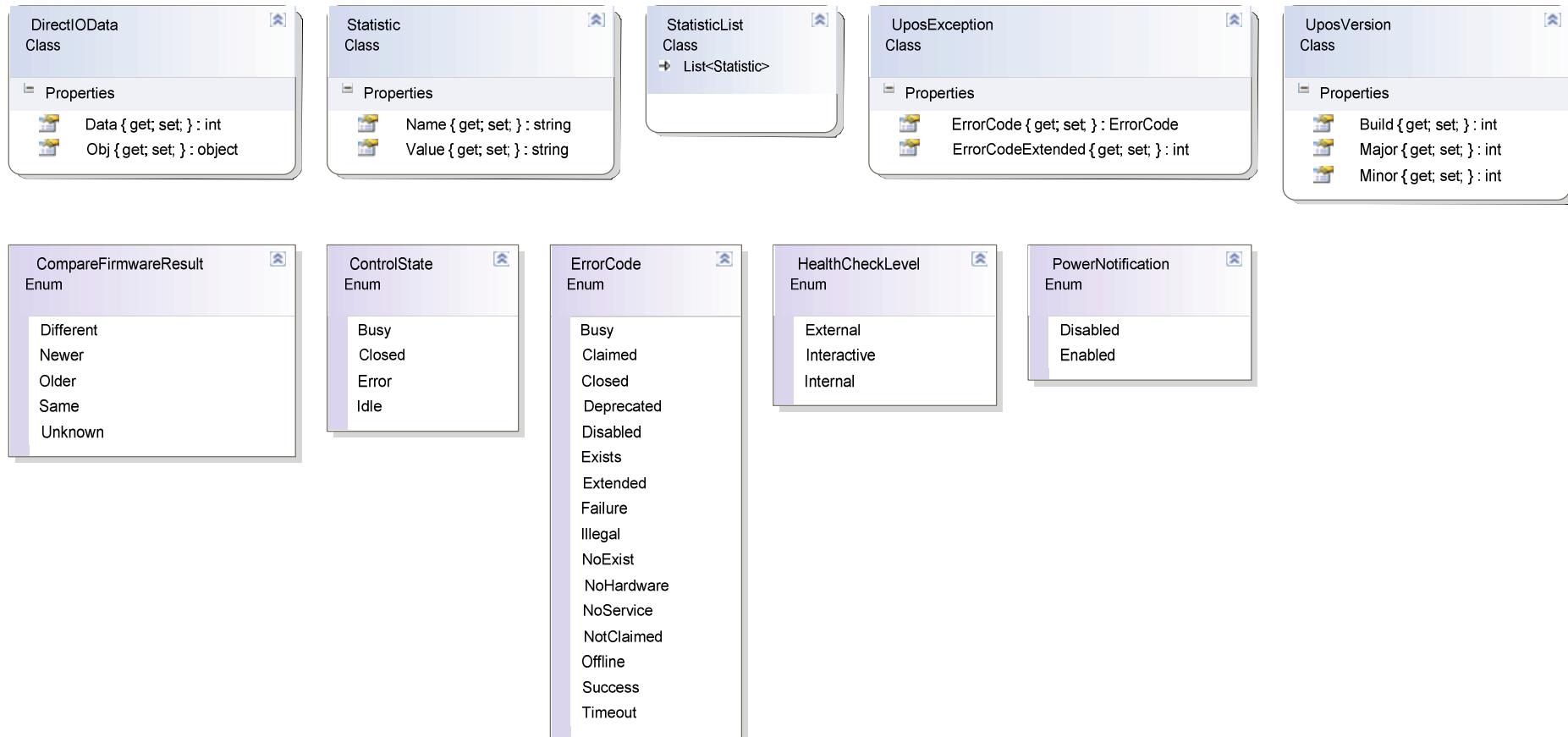
Smart Card Reader/Writer (Smart Card R/W)



Smart Card Reader/Writer (Smart Card R/W)



Smart Card Reader/Writer (Smart Card R/W)

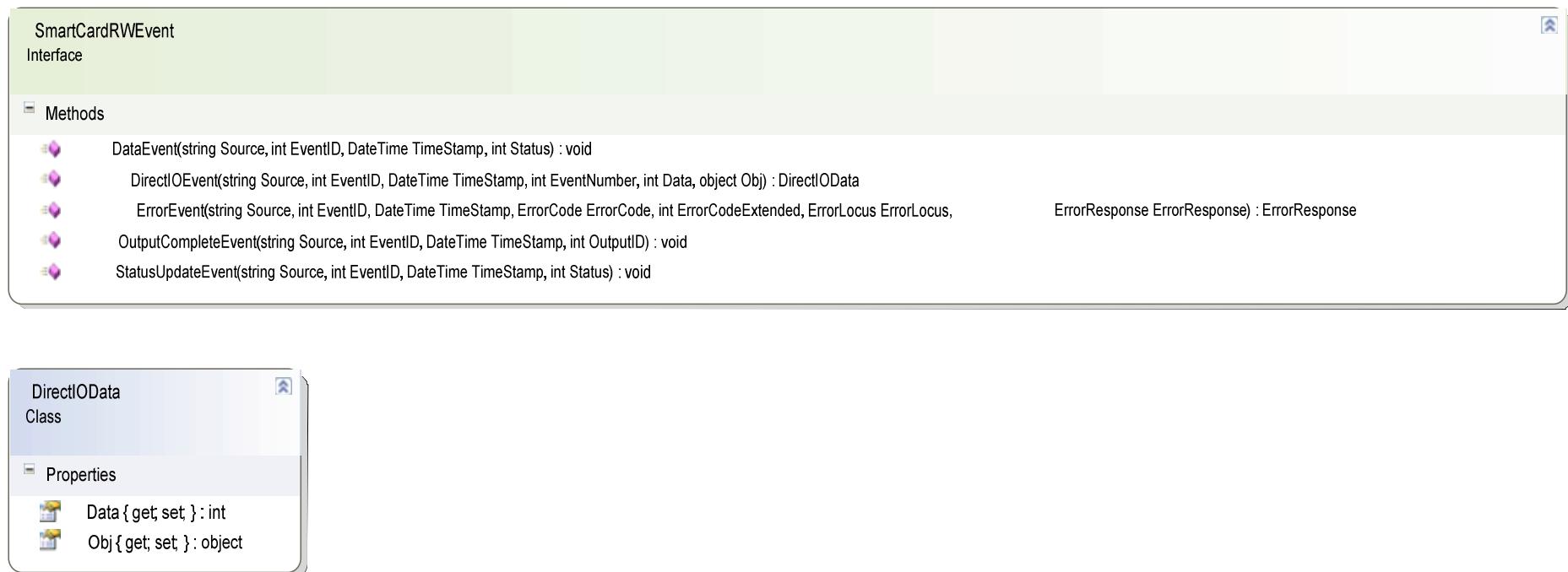


WS-POS 1.1 Technical Specification

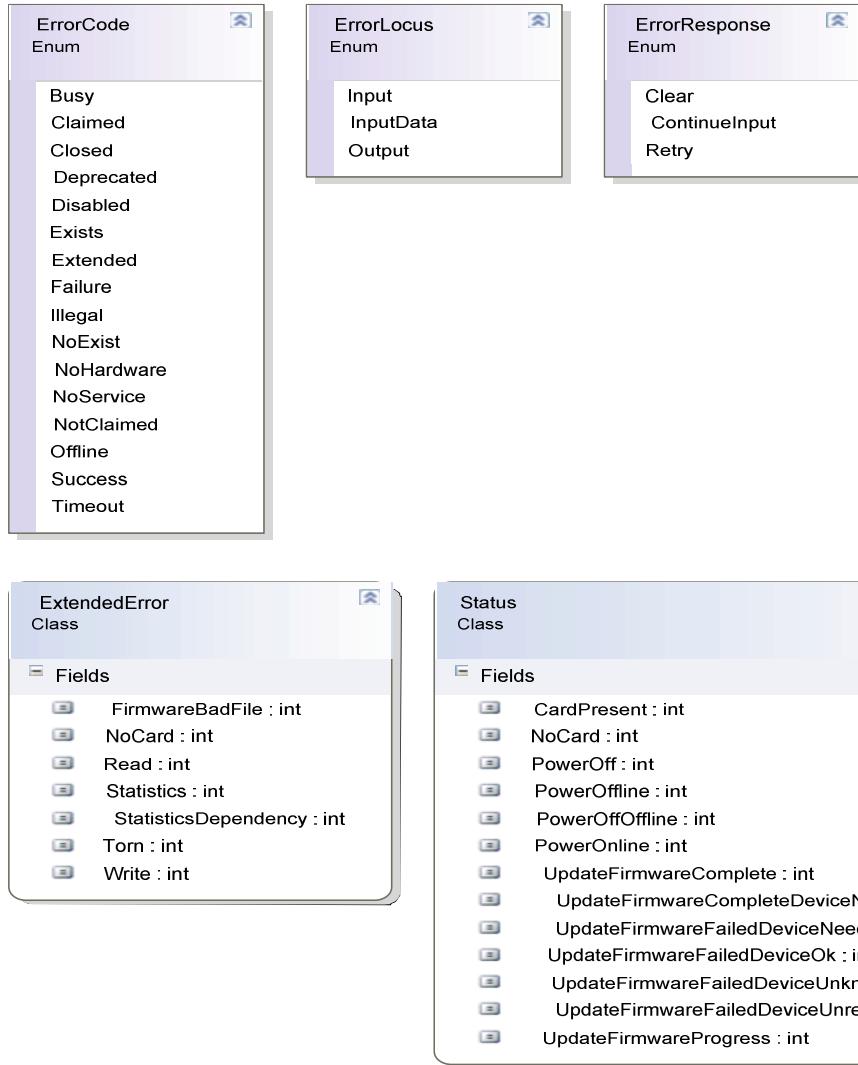
Smart Card Reader/Writer (Smart Card R/W)



Smart Card Reader/Writer (Smart Card R/W) Events



Smart Card Reader/Writer (Smart Card R/W) Events



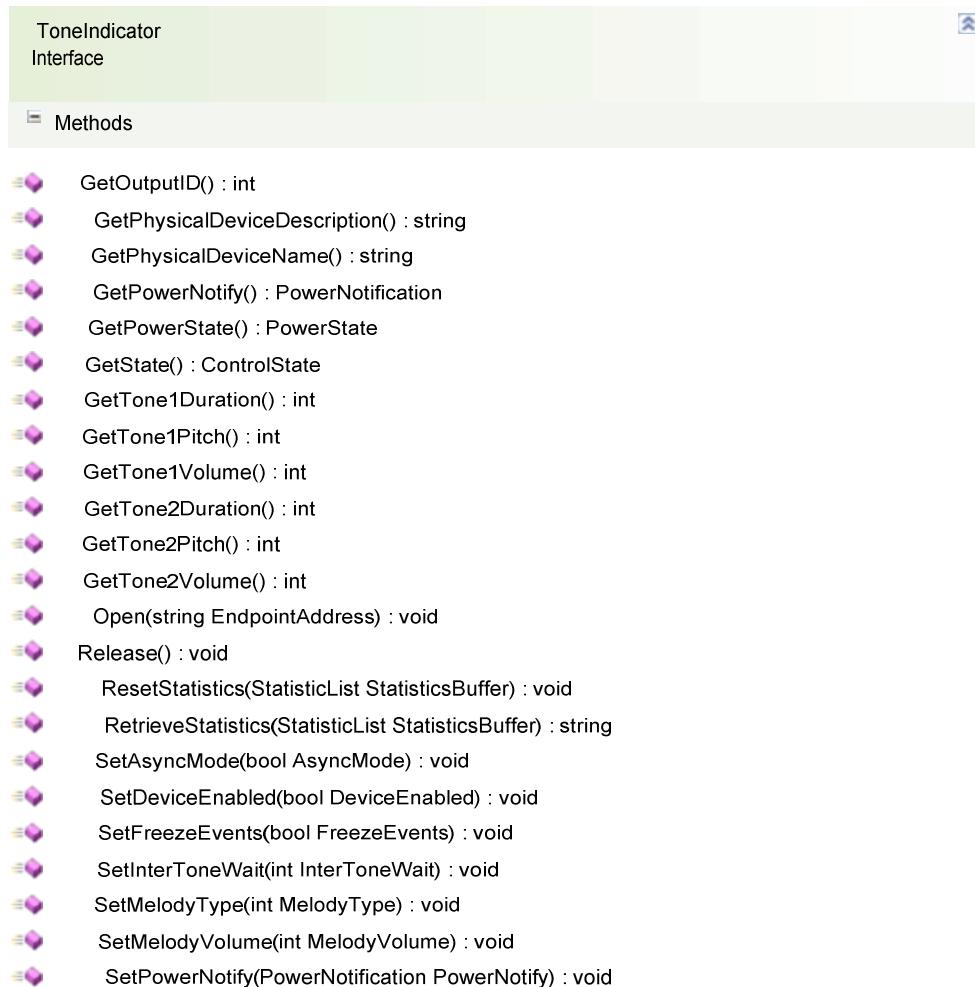
Tone Indicator

ToneIndicator
Interface

Methods

- CheckHealth(HealthCheckLevel Level) : void
- Claim(int Timeout) : void
- ClearOutput() : void
- Close(string EndpointAddress) : void
- CompareFirmwareVersion(string FirmwareFileName) : CompareFirmwareResult
- DirectIO(int Command, int Data, object Obj) : DirectIOData
- GetAsyncMode() : bool
- GetCapCompareFirmwareVersion() : bool
- GetCapMelody() : int
- GetCapPitch() : bool
 - GetCapPowerReporting() : PowerReporting
 - GetCapStatisticsReporting() : bool
 - GetCapUpdateFirmware() : bool
 - GetCapUpdateStatistics() : bool
 - GetCapVolume() : bool
 - GetCheckHealthText() : string
- GetClaimed() : bool
- GetDeviceControlDescription() : string
- GetDeviceControlVersion() : UposVersion
- GetDeviceEnabled() : bool
- GetDeviceServiceDescription() : string
- GetDeviceServiceVersion() : UposVersion
- GetFreezeEvents() : bool
- GetInterToneWait() : int
- GetMelodyType() : int
- GetMelodyVolume() : int

Tone Indicator

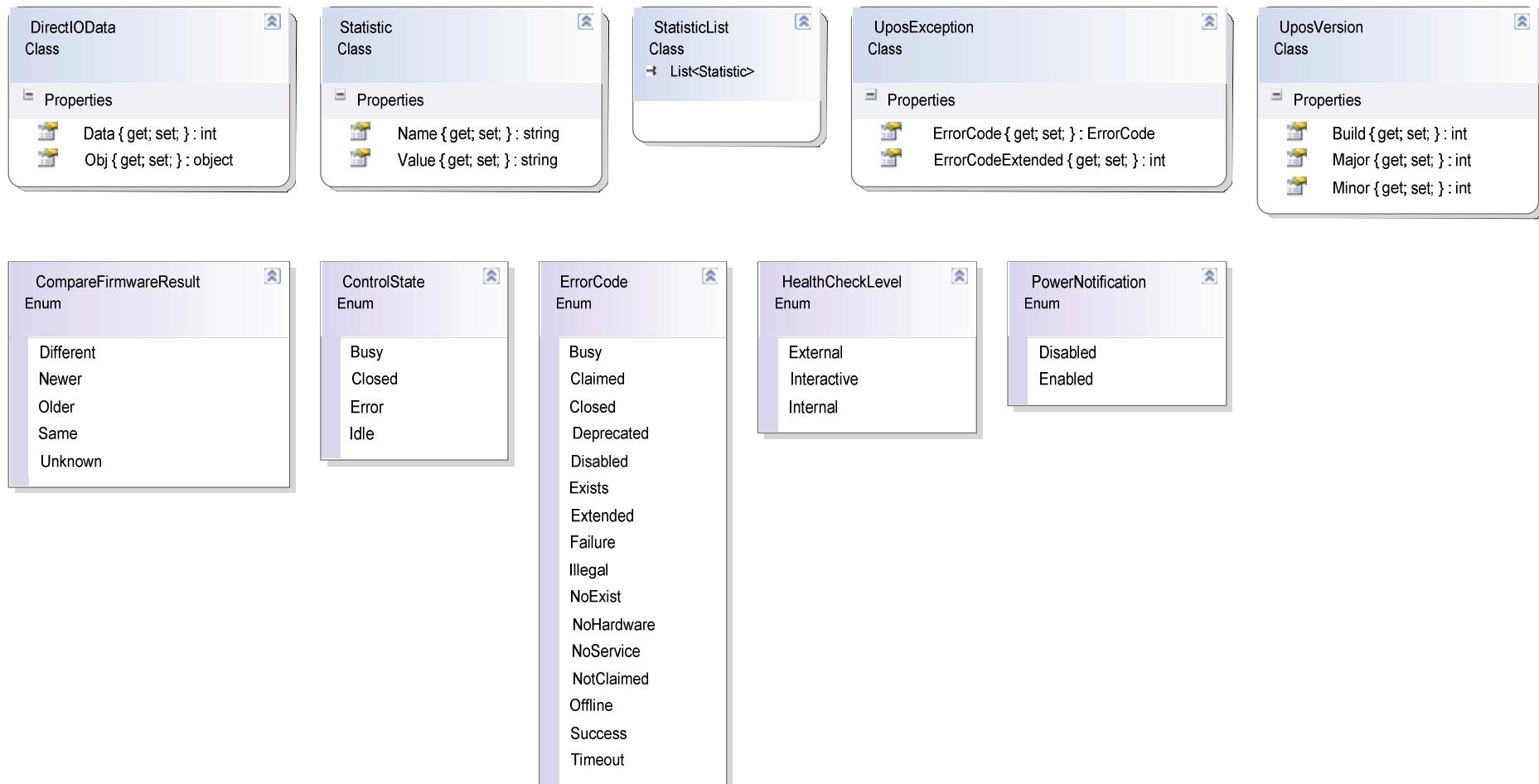


Tone Indicator

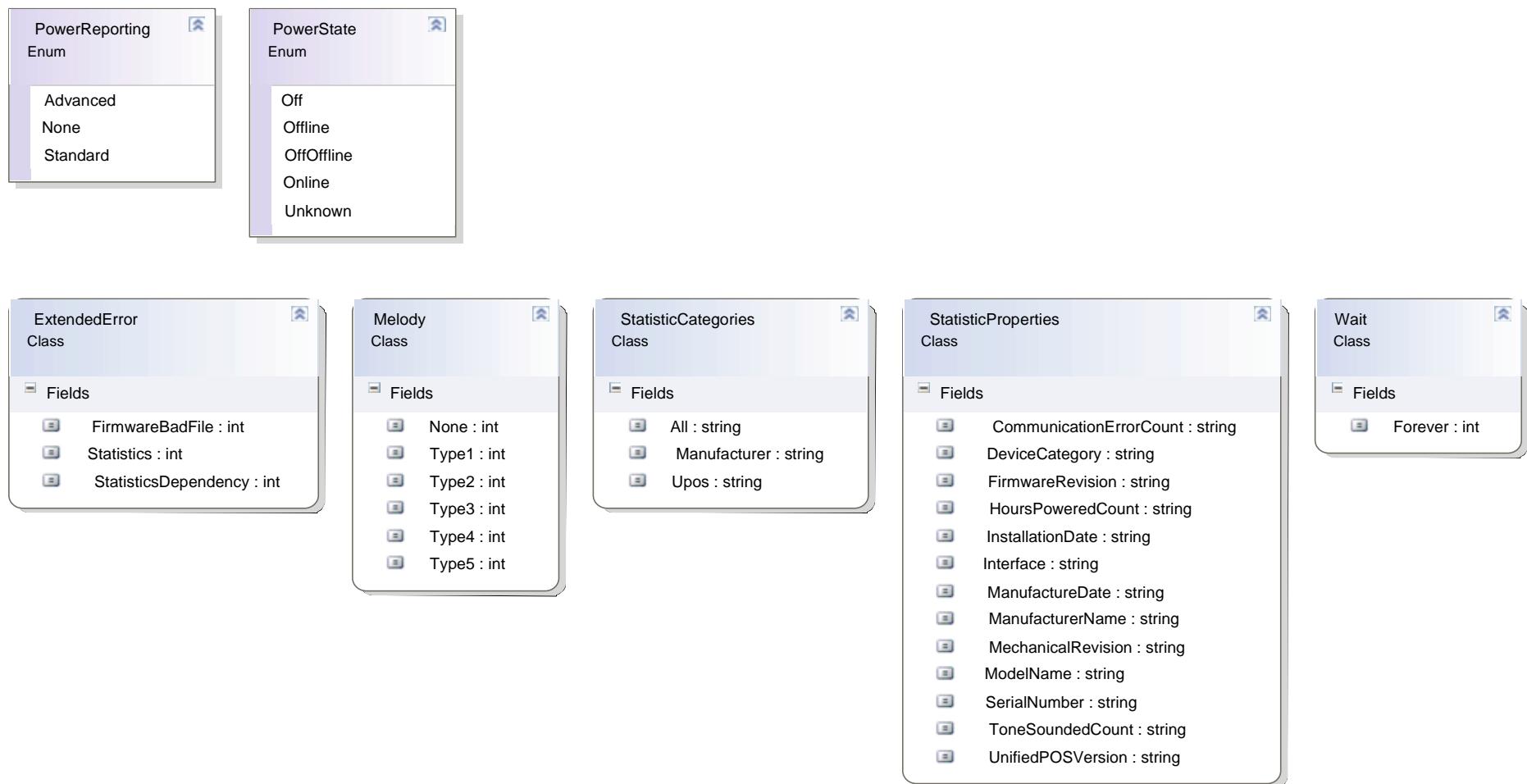


- SetTone1Duration(int Tone1Duration) : void
- SetTone1Pitch(int Tone1Pitch) : void
- SetTone1Volume(int Tone1Volume) : void
- SetTone2Duration(int Tone2Duration) : void
- SetTone2Pitch(int Tone2Pitch) : void
- SetTone2Volume(int Tone2Volume) : void
- Sound(int NumberOfCycles, int InterSoundWait) : void
- SoundImmediate() : void
- UpdateFirmware(string FirmwareFileName) : void
- UpdateStatistics(StatisticList StatisticsBuffer) : void

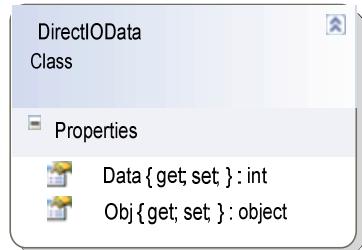
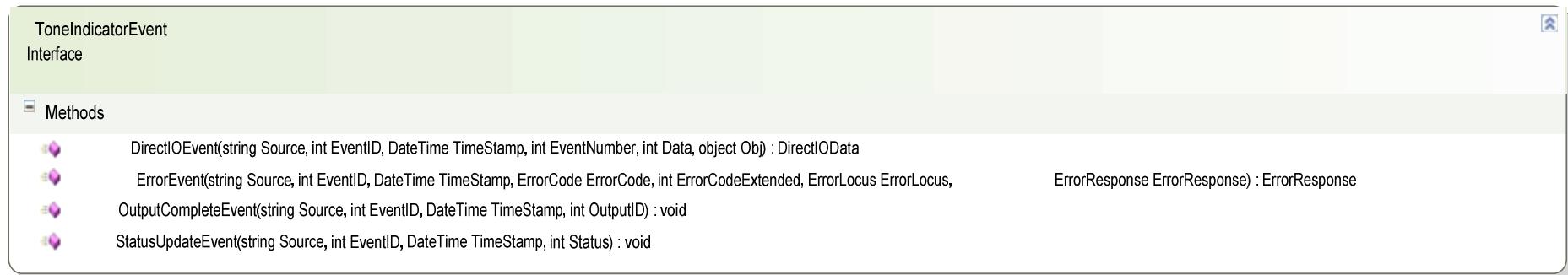
Tone Indicator



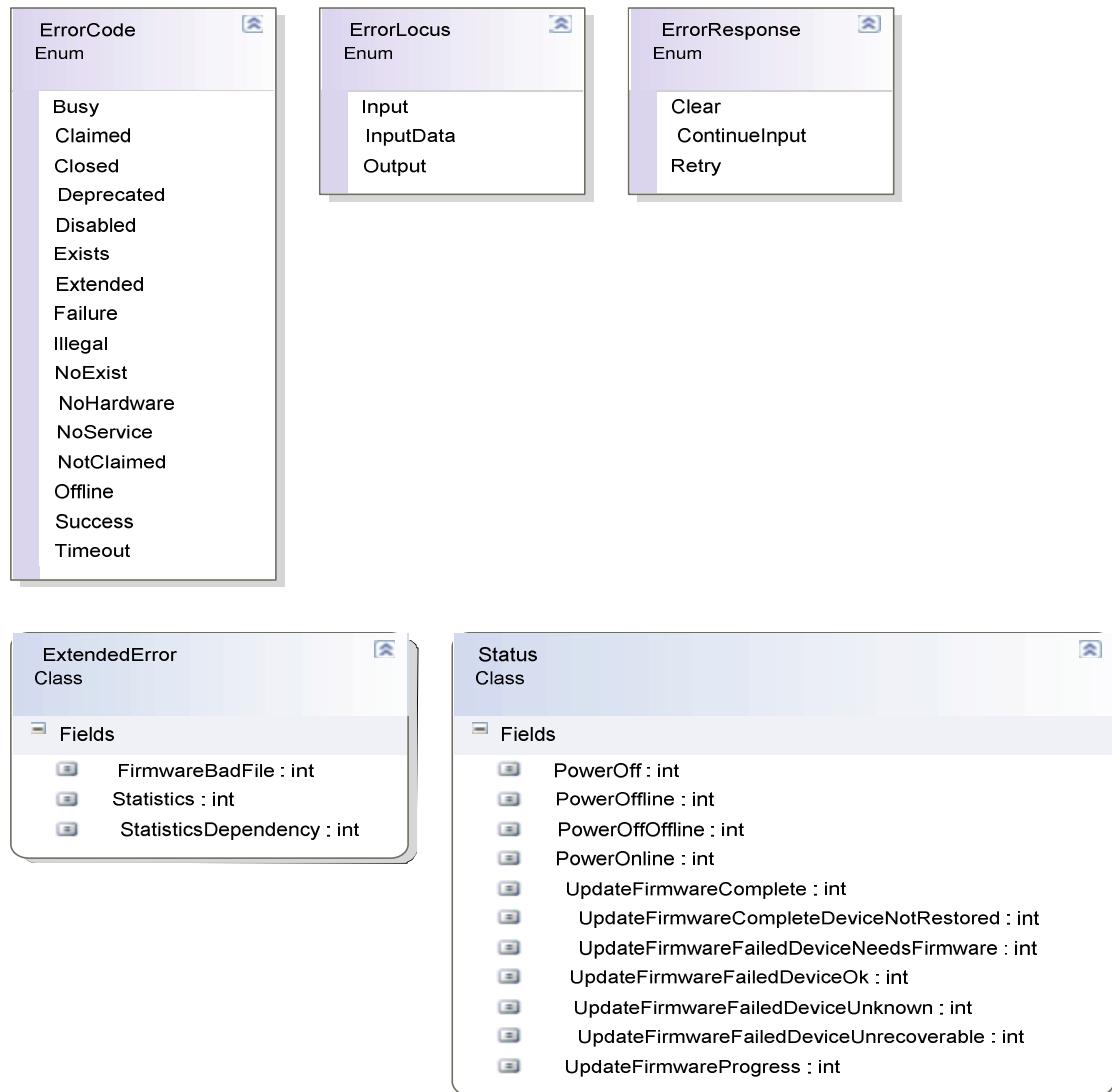
Tone Indicator



Tone Indicator Events



Tone Indicator Events



APPENDIX A APPLICATION DEVELOPMENT SUPPORT

The following provides an example of how a WS-POS mapping schema would work to resolve the UnifiedPOS Properties, Methods, and Events to WSDLs, C# web service contracts, and Java web service contracts.

First the example shows how the read-write property “DeviceEnable” and the method “checkHealth” for the UPOS device category “Belt” would be represented as WSDL document.

Second the corresponding Java and C# classes will be generated by using JAX-WS and WCF tools. All other properties, methods for the Belt are omitted (by ‘...’) for better readability.

Note that the Events are specified in separate WSDL files and are not included here in this example.

```
/*----- BeltV1.1.0.wsdl start -----*/
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns:wsap="http://schemas.xmlsoap.org/ws/2004/08/addressing/policy"
    xmlns: wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://www.nrf-arts.org/UnifiedPOS/Belt/"
    xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
    xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    xmlns:wsa10="http://www.w3.org/2005/08/addressing"
    xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
        name="BeltService"
        targetNamespace="http://www.nrf-arts.org/UnifiedPOS/Belt/"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
<wsdl:types>
    <xsd:schema targetNamespace="http://www.nrf-arts.org/UnifiedPOS/Belt/Imports">
        <xsd:import namespace="http://www.nrf-arts.org/UnifiedPOS/Belt/" schemaLocation="BeltV1.13.1.xsd" />
    </xsd:schema>
</wsdl:types>
...

```

WS-POS 1.1 Technical Specification

```
<wsdl:message name="Belt_GetDeviceEnabled_InputMessage">
  <wsdl:part name="parameters" element="tns:GetDeviceEnabled" />
</wsdl:message>

<wsdl:message name="Belt_GetDeviceEnabled_OutputMessage">
  <wsdl:part name="parameters" element="tns:GetDeviceEnabledResponse" />
</wsdl:message>

<wsdl:message name="Belt_GetDeviceEnabled_UposException_FaultMessage">
  <wsdl:part name="detail" element="tns:UposException" />
</wsdl:message>

<wsdl:message name="Belt_SetDeviceEnabled_InputMessage">
  <wsdl:part name="parameters" element="tns:SetDeviceEnabled" />
</wsdl:message>

<wsdl:message name="Belt_SetDeviceEnabled_OutputMessage">
  <wsdl:part name="parameters" element="tns:SetDeviceEnabledResponse" />
</wsdl:message>

<wsdl:message name="Belt_SetDeviceEnabled_UposException_FaultMessage">
  <wsdl:part name="detail" element="tns:UposException" />
</wsdl:message>

...
<wsdl:message name="Belt_CheckHealth_InputMessage">
  <wsdl:part name="parameters" element="tns:CheckHealth" />
</wsdl:message>

<wsdl:message name="Belt_CheckHealth_OutputMessage">
  <wsdl:part name="parameters" element="tns:CheckHealthResponse" />
</wsdl:message>

<wsdl:message name="Belt_CheckHealth_UposException_FaultMessage">
  <wsdl:part name="detail" element="tns:UposException" />
</wsdl:message>

...
<wsdl:portType name="Belt">
...
<wsdl:operation name="GetDeviceEnabled">

  <wsdl:input wsaw:Action="http://www.nrf-arts.org/UnifiedPOS/Belt/GetDeviceEnabled"
    message="tns:Belt_GetDeviceEnabled_InputMessage" />
```

WS-POS 1.1 Technical Specification

```
<wsdl:output wsaw:Action="http://www.nrf-arts.org/UnifiedPOS/Belt/GetDeviceEnabledResponse"  
message="tns:Belt_GetDeviceEnabled_OutputMessage" />  
  
<wsdl:fault wsaw:Action="http://www.nrf-arts.org/UnifiedPOS/Belt/UposException" name="UposException"  
message="tns:Belt_GetDeviceEnabled_UposException_FaultMessage" />  
  
</wsdl:operation>  
  
<wsdl:operation name="SetDeviceEnabled">  
  
    <wsdl:input wsaw:Action="http://www.nrf-arts.org/UnifiedPOS/Belt/SetDeviceEnabled"  
    message="tns:Belt_SetDeviceEnabled_InputMessage" />  
  
    <wsdl:output wsaw:Action="http://www.nrf-arts.org/UnifiedPOS/Belt/SetDeviceEnabledResponse"  
    message="tns:Belt_SetDeviceEnabled_OutputMessage" />  
  
    <wsdl:fault wsaw:Action="http://www.nrf-arts.org/UnifiedPOS/Belt/UposException" name="UposException"  
    message="tns:Belt_SetDeviceEnabled_UposException_FaultMessage" />  
  
</wsdl:operation>  
  
...  
  
    <wsdl:operation name="CheckHealth">  
  
        <wsdl:input wsaw:Action="http://www.nrf-arts.org/UnifiedPOS/Belt/CheckHealth"  
        message="tns:Belt_CheckHealth_InputMessage" />  
  
        <wsdl:output wsaw:Action="http://www.nrf-arts.org/UnifiedPOS/Belt/CheckHealthResponse"  
        message="tns:Belt_CheckHealth_OutputMessage" />  
  
        <wsdl:fault wsaw:Action="http://www.nrf-arts.org/UnifiedPOS/Belt/UposException" name="UposException"  
        message="tns:Belt_CheckHealth_UposException_FaultMessage" />  
  
    </wsdl:operation>  
  
...  
    </wsdl:portType>  
  
</wsdl:definitions>  
/*----- BeltV1.1.0.wsdl end -----*/
```

WS-POS 1.1 Technical Specification

Using the WCF tool “svccutil.exe” the following WCF contract in C# would be generated from:

```
/*----- Belt.cs start -----*/
using System;
using System.Collections.Generic;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace UnifiedPOS.Belt
{
    [ServiceContract(Namespace = "http://www.nrf-arts.org/UnifiedPOS/Belt/")]
    public interface Belt
    {
        ...
        [OperationContract(Action = "http://www.nrf-arts.org/UnifiedPOS/Belt/GetDeviceEnabled",
            ReplyAction = "http://www.nrf-arts.org/UnifiedPOS/Belt/GetDeviceEnabledResponse")]
        bool GetDeviceEnabled();
        [OperationContract(Action = "http://www.nrf-arts.org/UnifiedPOS/Belt/SetDeviceEnabled",
            ReplyAction = "http://www.nrf-arts.org/UnifiedPOS/Belt/SetDeviceEnabledResponse")]
        void SetDeviceEnabled(bool DeviceEnabled);
        ...
        [OperationContract(Action = "http://www.nrf-arts.org/UnifiedPOS/Belt/CheckHealth",
            ReplyAction = "http://www.nrf-arts.org/UnifiedPOS/Belt/CheckHealthResponse")]
        [FaultContract(typeof(UposException), Action = "http://www.nrf-arts.org/UnifiedPOS/Belt/UposException", Name = "UposException")]
        void CheckHealth(HealthCheckLevel Level);
        ...
    }
    ...
    [DataContract(Namespace = "http://www.nrf-arts.org/UnifiedPOS/Belt/")]
}
```

```
public enum HealthCheckLevel
{
    [EnumMember]
    External,
    [EnumMember]
    Interactive,
    [EnumMember]
    Internal,
}

...
/*----- Belt.cs end -----*/
```

For Java the JAX-WS contract generated with the JAX-WS tool “wsimport” would be look like the following:

```
/*----- Belt.java start -----*/
package org.nrf_arts.unifiedpos.belt;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;
import javax.xml.ws.Action;
import javax.xml.ws.FaultAction;
import javax.xml.ws.RequestWrapper;
import javax.xml.ws.ResponseWrapper;

@WebService(name = "Belt", targetNamespace = "http://www.nrf-arts.org/UnifiedPOS/Belt/")
public interface Belt
{
    ...
}
```

WS-POS 1.1 Technical Specification

```
@Action(input = "http://www.nrf-arts.org/UnifiedPOS/Belt/GetDeviceEnabled",
        output = "http://www.nrf-arts.org/UnifiedPOS/Belt/GetDeviceEnabledResponse",
        fault =
    {
        @FaultAction(className = org.nrf_arts.unifiedpos.belt.FaultException.class,
                    value = "http://www.nrf-arts.org/UnifiedPOS/Belt/UposException")
    })

@WebMethod(operationName = "GetDeviceEnabled",
           action = "http://www.nrf-arts.org/UnifiedPOS/Belt/GetDeviceEnabled")

@WebResult(name = "GetDeviceEnabledResult",
           targetNamespace = "http://www.nrf-arts.org/UnifiedPOS/Belt/")

@RequestWrapper(localName = "GetDeviceEnabled",
                targetNamespace = "http://www.nrf-arts.org/UnifiedPOS/Belt/",
                className = "org.nrf_arts.unifiedpos.belt.GetDeviceEnabled")

@ResponseWrapper(localName = "GetDeviceEnabledResponse",
                 targetNamespace = "http://www.nrf-arts.org/UnifiedPOS/Belt/",
                 className = "org.nrf_arts.unifiedpos.belt.GetDeviceEnabledResponse")

public Boolean getDeviceEnabled()
    throws FaultException;

@Action(input = "http://www.nrf-arts.org/UnifiedPOS/Belt/SetDeviceEnabled",
        output = "http://www.nrf-arts.org/UnifiedPOS/Belt/SetDeviceEnabledResponse",
        fault = {@FaultAction(className = org.nrf_arts.unifiedpos.belt.FaultException.class,
                            value = "http://www.nrf-arts.org/UnifiedPOS/Belt/UposException")
    })

@WebMethod(operationName = "SetDeviceEnabled",
           action = "http://www.nrf-arts.org/UnifiedPOS/Belt/SetDeviceEnabled")

@RequestWrapper(localName = "SetDeviceEnabled",
                targetNamespace = "http://www.nrf-arts.org/UnifiedPOS/Belt/",
                className = "org.nrf_arts.unifiedpos.belt.SetDeviceEnabled")

@ResponseWrapper(localName = "SetDeviceEnabledResponse",
                 targetNamespace = "http://www.nrf-arts.org/UnifiedPOS/Belt/",
                 className = "org.nrf_arts.unifiedpos.belt.SetDeviceEnabledResponse")

public void setDeviceEnabled(
    @WebParam(name = "DeviceEnabled",
              targetNamespace = "http://www.nrf-arts.org/UnifiedPOS/Belt/")

    Boolean deviceEnabled) throws FaultException;
```

```
...
@Action(input="http://www.nrf-arts.org/UnifiedPOS/Belt/CheckHealth",
        output="http://www.nrf-arts.org/UnifiedPOS/Belt/CheckHealthResponse",
        Fault={@FaultAction(className=org.nrf_arts.unifiedpos.belt.FaultException.class,
                           value="http://www.nrf-arts.org/UnifiedPOS/Belt/UposException")
              })
@WebMethod(operationName = "CheckHealth",
           action = "http://www.nrf-arts.org/UnifiedPOS/Belt/CheckHealth")

@RequestWrapper(localName = "CheckHealth",
                targetNamespace = "http://www.nrf-arts.org/UnifiedPOS/Belt/",
                className = "org.nrf_arts.unifiedpos.belt.CheckHealth")

@ResponseWrapper(localName = "CheckHealthResponse",
                 targetNamespace = "http://www.nrf-arts.org/UnifiedPOS/Belt/",
                 className = "org.nrf_arts.unifiedpos.belt.CheckHealthResponse")
public void checkHealth(
    @WebParam(name = "Level",
              targetNamespace = "http://www.nrf-arts.org/UnifiedPOS/Belt/")
    HealthCheckLevel level)
throws FaultException;

...
}

/*----- Belt.java end -----*/
```

Because JAX-WS has no possibility to declare a DataContract as in C#, for WSDL input and output actions, separate classes are generated providing the XML data access:

```
/*----- CheckHealth.java start -----*/
package org.nrf_arts.unifiedpos.belt;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;
```

WS-POS 1.1 Technical Specification

```
@XmlAccessorType(XmlAccessType.FIELD)  
  
@XmlType(name = "", propOrder = { "level" })  
  
@XmlRootElement(name = "CheckHealth")  
  
public class CheckHealth  
{  
  
    @XmlElement(name = "Level")  
    protected HealthCheckLevel level;  
  
    public HealthCheckLevel getLevel()  
    {  
        return level;  
    }  
  
    public void setLevel(HealthCheckLevel value)  
    {  
        this.level = value;  
    }  
}  
/*----- CheckHealth.java end -----*/
```

However, as the CheckHealth method does not return any data, the CheckHealthResponse class is empty:

```
/*----- CheckHealthResponse.java start -----*/  
package org.nrf_arts.unifiedpos.belt;  
  
import javax.xml.bind.annotation.XmlAccessType;  
import javax.xml.bind.annotation.XmlAccessorType;  
import javax.xml.bind.annotation.XmlRootElement;  
import javax.xml.bind.annotation.XmlType;  
  
@XmlAccessorType(XmlAccessType.FIELD)  
@XmlType(name = "")  
@XmlRootElement(name = "CheckHealthResponse")  
public class CheckHealthResponse  
{  
}  
/*----- CheckHealthResponse.java end -----*/  
/*-- Example End*/
```