Date: July 2019

OBJECT MANAGEMENT GROUP®

Unified POS RCSD

FTF Beta 1

This specification adds to and extends the UPOS 1.15 specification.

OMG Document Number: dtc/2019-07-04

Normative reference: https://www.omg.org/spec/UPOS/

This OMG document replaces the submission document (retail/2019-06-01, Alpha). It is an OMG Adopted Beta Specification and is currently in the finalization phase. Comments on the content of this document are welcome, and should be directed to issues@omg.org by October 25, 2019.

You may view the pending issues for this specification from the OMG revision issues web page https://issues.omg.org/issues/lists.

The FTF Recommendation and Report for this specification will be published in December 2019. If you are reading this after that date, please download the available specification from the OMG Specifications Catalog.

USE OF SPECIFICATION - TERMS, CONDITIONS & NOTICES

The material in this document details an Object Management Group specification in accordance with the terms, conditions and notices set forth below. This document does not represent a commitment to implement any portion of this specification in any company's products. The information contained in this document is subject to change without notice.

LICENSES

The companies listed above have granted to the Object Management Group, Inc. (OMG) a nonexclusive, royalty-free, paid up, worldwide license to copy and distribute this document and to modify this document and distribute copies of the modified version. Each of the copyright holders listed above has agreed that no person shall be deemed to have infringed the copyright in the included material of any such copyright holder by reason of having used the specification set forth herein or having conformed any computer software to the specification.

Subject to all of the terms and conditions below, the owners of the copyright in this specification hereby grant you a fully-paid up, non-exclusive, nontransferable, perpetual, worldwide license (without the right to sublicense), to use this specification to create and distribute software and special purpose specifications that are based upon this specification, and to use, copy, and distribute this specification as provided under the Copyright Act; provided that: (1) both the copyright notice identified above and this permission notice appear on any copies of this specification; (2) the use of the specifications is for informational purposes and will not be copied or posted on any network computer or broadcast in any media and will not be otherwise resold or transferred for commercial purposes; and (3) no modifications are made to this specification. This limited permission automatically terminates without notice if you breach any of these terms or conditions. Upon termination, you will destroy immediately any copies of the specifications in your possession or control.

PATENTS

The attention of adopters is directed to the possibility that compliance with or adoption of OMG specifications may require use of an invention covered by patent rights. OMG shall not be responsible for identifying patents for which a license may be required by any OMG specification, or for conducting legal inquiries into the legal validity or scope of those patents that are brought to its attention. OMG specifications are prospective and advisory only. Prospective users are responsible for protecting themselves against liability for infringement of patents.

IPR Mode

This specification is issued under the RAND Mode based on the OMG IPR Policy.
OMG IPR Policy https://www.omg.org/cgi-bin/doc.cgi?ipr

GENERAL USE RESTRICTIONS

Any unauthorized use of this specification may violate copyright laws, trademark laws, and communications regulations and statutes. This document contains information which is protected by copyright. All Rights Reserved. No part of this work covered by copyright herein may be reproduced or used in any form or by any means--graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems--without permission of the copyright owner.

DISCLAIMER OF WARRANTY

WHILE THIS PUBLICATION IS BELIEVED TO BE ACCURATE, IT IS PROVIDED "AS IS" AND MAY CONTAIN ERRORS OR MISPRINTS. THE OBJECT MANAGEMENT GROUP AND THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF TITLE OR OWNERSHIP, IMPLIED WARRANTY OF MERCHANTABILITY OR WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR USE. IN NO EVENT SHALL THE OBJECT MANAGEMENT GROUP OR ANY OF THE COMPANIES LISTED ABOVE BE LIABLE FOR ERRORS CONTAINED HEREIN OR FOR DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL, RELIANCE OR COVER DAMAGES, INCLUDING LOSS OF PROFITS, REVENUE, DATA OR USE, INCURRED BY ANY USER OR ANY THIRD PARTY IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS MATERIAL, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The entire risk as to the quality and performance of software developed using this specification is borne by you. This disclaimer of warranty constitutes an essential part of the license granted to you to use this specification.

RESTRICTED RIGHTS LEGEND

Use, duplication or disclosure by the U.S. Government is subject to the restrictions set forth in subparagraph (c) (1) (ii) of The Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clauses at 48 C.F.R. 52.227-19 or as specified in 48 C.F.R. 227-7202-2 of the DoD F.A.R. Supplement and its successors, or as specified in 48 C.F.R. 12.212 of the Federal Acquisition Regulations and its successors, as applicable. The specification copyright owners are as indicated above and may be contacted through the Object Management Group, 109 Highland Avenue, Needham, MA 02494, U.S.A.

TRADEMARKS

CORBA®, CORBA logos®, FIBO®, Financial Industry Business Ontology®, FINANCIAL INSTRUMENT GLOBAL IDENTIFIER®, IIOP®, IMM®, Model Driven Architecture®, MDA®, Object Management Group®, OMG®, OMG Logo®, SoaML®, SOAML®, SysML®, UAF®, Unified Modeling Language®, UML®, UML Cube Logo®, VSIPL®, and XMI® are registered trademarks of the Object Management Group, Inc.

For a complete list of trademarks, see: https://www.omg.org/legal/tm_list.htm. All other products or company names mentioned are used for identification purposes only, and may be trademarks of their respective owners.

COMPLIANCE

The copyright holders listed above acknowledge that the Object Management Group (acting itself or through its designees) is and shall at all times be the sole entity that may authorize developers, suppliers and sellers of computer software to use certification marks, trademarks or other special designations to indicate compliance with these materials.

Software developed under the terms of this license may claim compliance or conformance with this specification if and only if the software compliance is of a nature fully matching the applicable compliance points as stated in the specification. Software developed only partially matching the applicable compliance points may claim only that the software was based on this specification, but may not claim compliance or conformance with this specification. In the event that testing suites are implemented or approved by Object Management Group, Inc., software developed using this specification may claim compliance or conformance with the specification only if the software satisfactorily completes the testing suites.

OMG's Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page https://www.omg.org, under Documents, Report a Bug/Issue.

Document Sumbitter
Vinx Corp.

Document Publishing Supportes

OPOS-J
Sorimachi Giken Co. Ltd.
Microsoft Japan Ltd.
SEIKO EPSON Corp.
Toshiba TEC Corp.
Star Micronics Corp.
Fujitsu Frontec Corp.
NCR Corporation
Sharp Corporation
Omron Social Solutions Corp.
NEC Platforms Corp.
Transaction Media Netwo
rks Inc.

Table of Content

Preface

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at https://www.omg.org/.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. All OMG Specifications are available from the OMG website at:

https://www.omg.org/spec

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
109 Highland Avenue
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: *pubs@omg.org*

Certain OMG specifications are also available as ISO standards. Please consult http://www.iso.org

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

NOTE: Terms that appear in italics are defined in the glossary. Italic text also represents the name of a document, specification, or other publication.

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to https://www.omg.org/report_issue.htm.

UPOS 1.16 RCSD Specification Overivew

Updated Items in Release 1.16

Chapter sections 23 and 38 from UPOS1.15 are included with annotations denoting the changes necessary for supporting the addition of the Retail Communications Service Devices. Chapters 39-47 are new chapters for devices being added to UPOS v1. The following is a list of the proterties, methods and chapters.

Updated Items in CHAPTER 21 Lights

Properties

CapFullColor **Property**
CapPattern **Property**
FullColor **Property**

Methods

switchOn **Method**
switchONMultiple **Method**
switchOnPattern **Method**
switchOffPattern **Method**

Updated Items in CHAPTER 29 POS Power

Properties

CapChargeTime **Property**
CapTimeMode **Property**
ChargeTime **Property**
TimeMode **Property**

Added Chapters in Release 1.16

CHAPTER 39 Video Capture

CHAPTER 40 Individual Recognition

CHAPTER 41 Sound Recorder

CHAPTER 42 Voice Recognition

CHAPTER 43 Sound Player

CHAPTER 44 Speech Synthesis

CHAPTER 45 Gesture Control

CHAPTER 46 Device Monitor

CHAPTER 47 Graphic Display

CHAPTER 21

Lights

This Chapter defines the Lights device category.

Summary

Properties (UML attributes)

Common	*Type*	*Mutability*	*Version*	*May Use After*
AutoDisable:	*boolean*	{read-write}	1.12	Not Supported
CapCompareFirmwareVersion:	*boolean*	{read-only}	1.12	open
CapPowerReporting:	*int32*	{read-only}	1.12	open
CapStatisticsReporting:	*boolean*	{read-only}	1.12	open
CapUpdateFirmware:	*boolean*	{read-only}	1.12	open
CapUpdateStatistics:	*boolean*	{read-only}	1.12	open
CheckHealthText:	*string*	{read-only}	1.12	open
Claimed:	*boolean*	{read-only}	1.12	open
DataCount:	*int32*	{read-only}	1.12	Not Supported
DataEventEnabled:	*boolean*	{read-write}	1.12	Not Supported
DeviceEnabled:	*boolean*	{read-write}	1.12	open, claim
FreezeEvents:	*boolean*	{read-write}	1.12	open
OutputID:	*int32*	{read-only}	1.12	Not Supported
PowerNotify:	*int32*	{read-write}	1.12	open
PowerState:	*int32*	{read-only}	1.12	open
State:	*int32*	{read-only}	1.12	--
DeviceControlDescription:	*string*	{read-only}	1.12	--
DeviceControlVersion:	*int32*	{read-only}	1.12	--
DeviceServiceDescription:	*string*	{read-only}	1.12	open
DeviceServiceVersion:	*int32*	{read-only}	1.12	open
PhysicalDeviceDescription:	*string*	{read-only}	1.12	open
PhysicalDeviceName:	*string*	{read-only}	1.12	open

Properties (Continued)

Specific	*Type*	*Mutability*	*Version*	*May Use After*
CapAlarm:	*int32*	{read-only}	1.12	open
CapBlink:	*boolean*	{read-only}	1.12	open
CapColor:	*int32*	{read-only}	1.12	open
CapFullColor:	*boolean*	{read-only}	1.16	open
CapPattern:	*int32*	{read-only}	1.16	open
FullColor:	*boolean*	{read-write}	1.16	open
MaxLights:	*int32*	{read-only}	1.12	open

Methods (UML operations)

Common

Name	*Version*
open (logicalDeviceName: *string*`)`**:** **void {raises-exception}**	1.12
close ( ): **void {raises-exception, use after open}**	1.12
claim (timeout: *int32*`)`**:** **void {raises-exception, use after open}**	1.12
release ( ): **void {raises-exception, use after open, claim}**	1.12
checkHealth (level: *int32*`)`**:** **void {raises-exception, use after open, enable}**	1.12
clearInput ( ): **void { }**	***Not supported***
clearInputProperties ( ): **void { }**	***Not supported***
clearOutput ( ): **void { }**	***Not supported***
directIO (command: *int32*`,` **inout data:** *int32*`,` **inout obj:** *object*`)`**:** **void {raises-exception, use after open}**	1.12
compareFirmwareVersion (firmwareFileName: *string*`,` **out result:** *int32*`)`**:** **void {raises-exception, use after open, enable}**	1.12
resetStatistics (statisticsBuffer: *string*`)`**:** **void {raises-exception, use after open, enable}**	1.12

retrieveStatistics (inout statisticsBuffer: *string*): 1.12
void {raises-exception, use after open, enable}

updateFirmware (firmwareFileName: *string*): 1.12
void {raises-exception, use after open, enable}

updateStatistics (statisticsBuffer: *string*): 1.12
void {raises-exception, use after open, enable}

Specific

Name

switchOff (lightNumber*: int32*): 1.12
void {raises-exception, use after open, claim, enable}

switchOn (lightNumber: *int32*, **blinkOnCycle:** *int32*, 1.16
blinkOffCycle: *int32*, **color:** *int32*, **alarm:** *int32*):
void {raises-exception, use after open, claim, enable}

switchOnMultiple (lightNumbers: string, blinkOnCycle: int32, 1.16
blinkOffCycle: int32, color: int32, alarm: int32):
void {raises-exception, use after open, claim, enable}

switchOnPattern (pattern: int32, alarm: int32): 1.16
void {raises-exception, use after open, claim, enable}

switchOffPattern ( ): 1.16
void {raises-exception, use after open, claim, enable}

Events (UML interfaces)

Name	*Type*	*Mutability*	*Version*
upos::events::DataEvent		***Not Supported***	
upos::events::DirectIOEvent			1.12
EventNumber:	*int32*	{read-only}	
Data:	*int32*	{read-write}	
Obj:	*object*	{read-write}	
upos::events::ErrorEvent		***Not Supported***	
upos::events::OutputCompleteEvent		***Not Supported***	
upos::events::StatusUpdateEvent			1.12
Status:	*int32*	{read-only}	

General Information

The Lights programmatic name is “Lights”.

This device category was added to Version 1.12 of the specification.

Capabilities

- The Lights Control has the following capability:
  - Supports commands to “switch on” and “switch off” a light.
- The Lights Control may have the following additional capabilities:
  - Supports device-level blinking at adjustable blink cycles.
  - Support multiple lights.
  - Supports different colors of a light.
  - Supports different alarms

Device Sharing

Lights is an exclusive-use device. Its device sharing rules are:

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

Lights Class Diagram

Updated in Release 1.16

The following diagram shows the relationships between the Lights classes

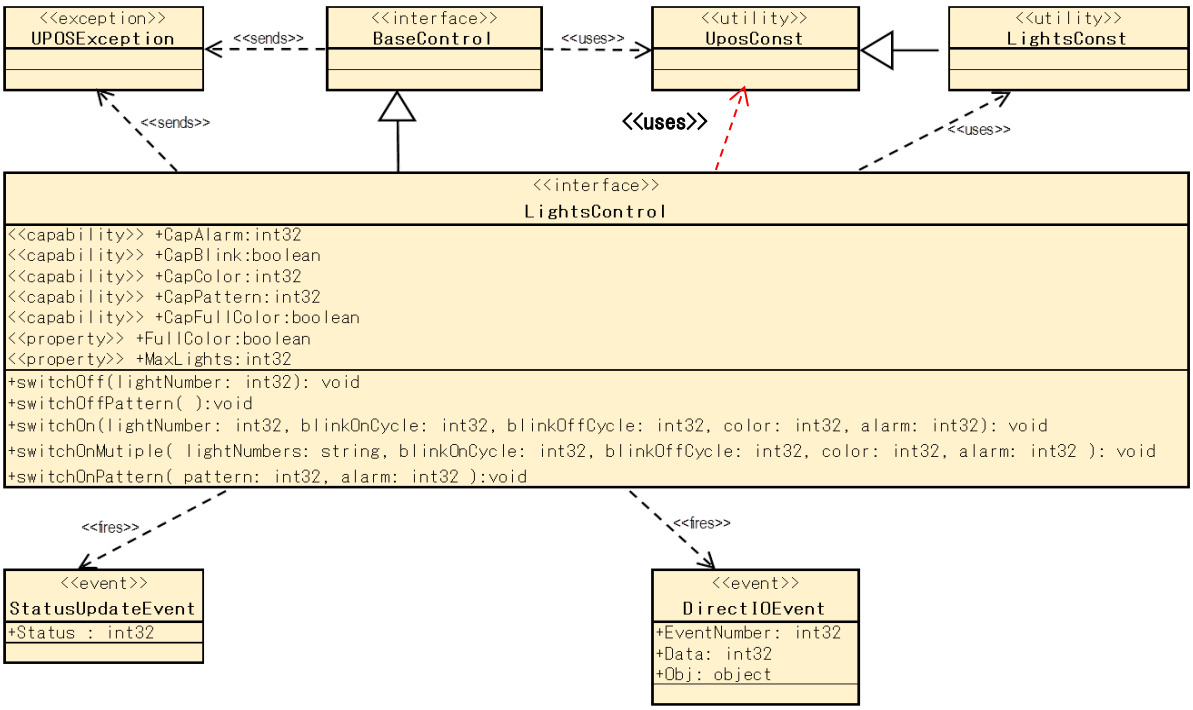


Fig. Chap. 21-1 Lights Class Diagram

Lights Sequence Diagram

The following sequence diagram show the typical usage of the Lights device illustrating the handling of the media entry indicator lights.

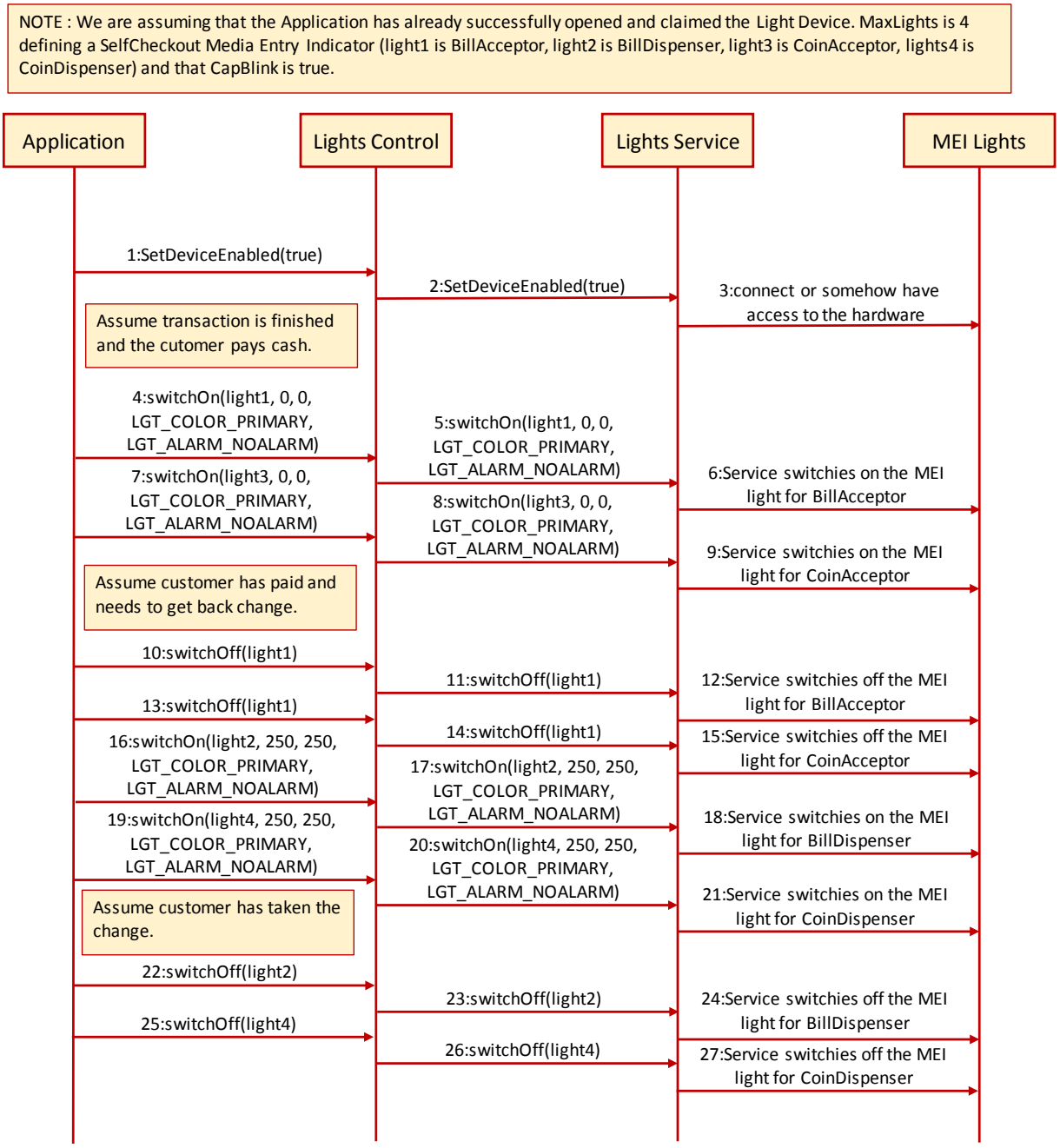


Fig. Chap. 21-2 Lights Sequence Diagram (handling of the media entry indicator lights)

The following sequence diagram show the typical usage of the Lights device illustrating the handling of the pole lights.

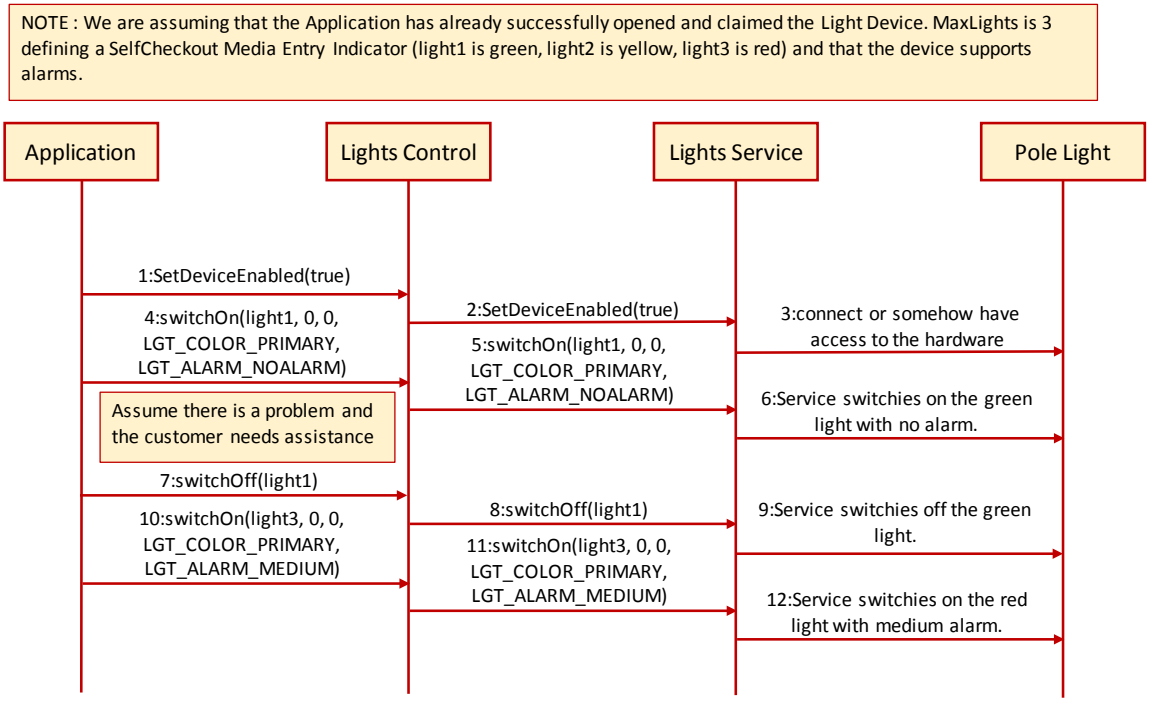


Fig. Chap. 21-3 Lights Sequence Diagram (handling of the pole lights)

Properties (UML attributes)

CapAlarm Property

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

Remarks This capability indicates if the device supports different alarms.

CapAlarm is a logical OR combination of any of the following values:

Value	Meaning
LGT_ALARM_NOALARM	Alarms are not supported.
LGT_ALARM_SLOW	Supports a slow beep.
LGT_ALARM_MEDIUM	Supports a medium beep.
LGT_ALARM_FAST	Supports a fast beep.
LGT_ALARM_CUSTOM1	Supports 1st custom alarm.
LGT_ALARM_CUSTOM2	Supports 2nd custom alarm.

This property is initialized by the **open** method. If the device does not support alarms, it is initialized to LGT_ALARM_NOALARM.

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

CapBlink Property

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

Remarks If true, a blinking capability is supported. It may be either a physical capability of the device or emulated by the service.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

CapColor Property

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

Remarks This capability indicates if the device supports different colors.

CapColor is a logical OR combination of any of the following values:

Value	Meaning
LGT_COLOR_PRIMARY	Supports Primary Color (Usually Green).
LGT_COLOR_CUSTOM1	Supports 1st Custom Color (Usually Red).
LGT_COLOR_CUSTOM2	Supports 2nd Custom Color (Usually Yellow).
LGT_COLOR_CUSTOM3	Supports 3rd Custom Color.
LGT_COLOR_CUSTOM4	Supports 4th Custom Color.
LGT_COLOR_CUSTOM5	Supports 5th Custom Color.

This property is initialized by the **open** method. If the device supports only one color, it is initialized to LGT_COLOR_PRIMARY.

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

CapFullColor Property *Added in Release 1.16*

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

Remarks If true, the application can set **FullColor** property to true and specify full color.

If false, the application cannot specify full color.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **FullColor** Property, **switchOn** Method, **switchOnMultiple** Method.

CapPattern Property *Added in Release 1.16*

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

Remarks This capability indicates if the device supports different lighting patterns.

CapPattern is a logical OR combination of any of the following values:

Value	Meaning
LGT_PATTERN_NOPATTERN	Lighting patterns are not supported.
LGT_ PATTERN _CUSTOM	1~32 Supports 1st to 32th Lighting Pattern.

This property is initialized by the **open** method. If the device does not support lighting pattern, it is initialized to LGT_PATTERN_NOPATTERN.

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **switchOnPattern** Method.

FullColor Property *Added in Release 1.16*

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

Remarks Holds the format of the value to specify for the *Color* parameter of **SwitchOn** method and **SwitchOnMultiple** method.

If true, the *Color* parameter format is full color of 0xRRGGBB format.

If false, the *Color* parameter format is one of the colors defined by **CapColor**.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **CapFullColor** Property, **switchOn** Method, **switchOnMultiple** Method.

MaxLights Property

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

Remarks **MaxLights** specifies the maximum number of lights that the device can support.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

Methods (UML operations)

switchOff Method

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

Parameter	Description
lightNumber	Specifies the light number. Valid light numbers are 1 through **MaxLights**.

Remarks Switches off the light specified by *lightNumber*.

Errors A UposException may be thrown when this method is invoked. For further information, see **"Errors"** on page Intro-20**.**

A possible value of the exception's *ErrorCode* property is:

Value	Meaning
E_ILLEGAL	The *lightNumber* parameter exceeds **MaxLights**.

See Also **MaxLights** Property**.**

switchOffPattern Method

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

Remarks Switches off the pattern lighting.

Errors A UposException may be thrown when this method is invoked. For further information, see **"Errors"** on page Intro-20**.**

A possible value of the exception's *ErrorCode* property is:

Value	Meaning
E_ILLEGAL	Pattern lighting is not executed.

See Also **switchOnPattern** Method.

switchOn Method *Updated in Release 1.16*

Syntax **switchOn (lightNumber: *int32,* blinkOnCycle: *int32,* blinkOffCycle: *int32,* color: *int32,* alarm: *int32*): void {raises-exception, use after open, claim, enable}**

Parameter	Description
lightNumber	Specifies the light number. Valid light numbers are 1 through **MaxLights**.
blinkOnCycle	A zero (0) value indicates no blink cycle. A positive value indicates the blink on cycle time in milliseconds. Negative values are not allowed.
blinkOffCycle	A zero (0) value indicates no blink cycle. A positive value indicates the blink off cycle time in milliseconds. Negative values are not allowed.
color	If **FullColor** is true, specifies the color of the light, must be full color of 0xRRGGBB format. If **FullColor** is false, specifies the color of the light, must be one of the colors defined by **CapColor**.
alarm	Specifies the used alarm type, must be one of the alarms defined by **CapAlarm**.

Remarks Switches on the light specified by *lightNumber* or let it blink.

If *blinkOnCycle* and *blinkOffCycle* are zero (0) or **CapBlink** is false, then the parameters *blinkOnCycle* and *blinkOffCycle* will be ignored and the light will only be switched on.

If **CapBlink** is true and *blinkOnCycle* and *blinkOffCycle* are positive then the light will blink.

If **CapColor** is LGT_COLOR_PRIMARY the light does not support different colors and *color* is ignored, otherwise **switchOn** will use the color specified by *color*.

If **CapAlarm** is LGT_ALARM_NOALARM the light does not support different alarms and *alarm* is ignored, otherwise **switchOn** will use the alarm specified by *alarm*.

Subsequent calls to **switchOn** will change the blink cycles, the color or the alarm type of the light.

Errors A UposException may be thrown when this method is invoked. For further information, see **"Errors"** on page Intro-20**.**

A possible value of the exception's *ErrorCode* property is:

Value	Meaning
E_ILLEGAL	The *lightNumber* parameter exceeds **MaxLights**, an invalid *color* or *alarm* was specified.

See Also **CapAlarm** Property, **CapBlink** Property, **CapColor** Property, **FullColor** Property, **MaxLights** Property**.**

switchOnMultiple Method *Added in Release 1.16*

Syntax **switchOnMultiple (lightNumbers: *string*, blinkOnCycle: *int32*, blinkOffCycle: *int32*, color: *int32*, alarm: *int32*): void {raises-exception, use after open, claim, enable}**

Parameter	Description
lightNumbers	Specifies the comma-delimited list of light number. Valid light numbers are 1 through **MaxLights**.
blinkOnCycle	A zero (0) value indicates no blink cycle. A positive value indicates the blink on cycle time in milliseconds. Negative values are not allowed.
blinkOffCycle	A zero (0) value indicates no blink cycle. A positive value indicates the blink off cycle time in milliseconds. Negative values are not allowed.
color	If **FullColor** is true, specifies the color of the light, must be full color of 0xRRGGBB format. If **FullColor** is false, specifies the color of the light, must be one of the colors defined by **CapColor**.
alarm	Specifies the used alarm type, must be one of the alarms defined by **CapAlarm**.

Remarks Switches on the multiple lights specified by *lightNumbers* or let it blink.

If *blinkOnCycle* and *blinkOffCycle* are zero (0) or **CapBlink** is false, then the parameters *blinkOnCycle* and *blinkOffCycle* will be ignored and the light will only be switched on.

If **CapBlink** is true and *blinkOnCycle* and *blinkOffCycle* are positive then the light will blink.

If **CapColor** is LGT_COLOR_PRIMARY the light does not support different colors and *color* is ignored, otherwise **switchOnMultiple** will use the color specified by *color*.

If **CapAlarm** is LGT_ALARM_NOALARM the light does not support different alarms and *alarm* is ignored, otherwise **switchOnMultiple** will use the alarm specified by *alarm*.

Errors A UposException may be thrown when this method is invoked. For further information, see **"Errors"** on page Intro-20**.**

A possible value of the exception's *ErrorCode* property is:

Value	Meaning
E_ILLEGAL	The *lightNumbers* parameter exceeds **MaxLights**, an invalid value was specified.

See Also **CapAlarm** Property, **CapBlink** Property, **CapColor** Property, **FullColor** Property, **MaxLights** Property**.**

switchOnPattern Method *Added in Release 1.16*

Syntax **switchOnPattern (pattern: *int32*, alarm: *int32*):**
void {raises-exception, use after open, claim, enable}

Parameter	Description
pattern	Specifies the lighting pattern, must be one of the pattern defined by **CapPattern**.
alarm	Specifies the used alarm type, must be one of the alarms defined by **CapAlarm**.

Remarks Switches on the light specified by *pattern*.

If **CapAlarm** is LGT_ALARM_NOALARM the light does not support different alarms and *alarm* is ignored, otherwise **switchOn** and **swithOnPattern** will use the alarm specified by *alarm*.

Errors A UposException may be thrown when this method is invoked. For further information, see **"Errors"** on page Intro-20**.**

A possible value of the exception's *ErrorCode* property is:

Value	Meaning
E_ILLEGAL	An invalid value was specified, or unsupported operation with the Device.

See Also **CapAlarm** Property, **CapPattern** Property**.**

Events (UML interfaces)

DirectIOEvent

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

EventNumber:	***int32*** **{read-only}**
Data:	***int32*** **{read-write}**
Obj:	***object*****{read-write}**

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

Attributes This event contains the following attributes:

Attribute	Type	Description
EventNumber	*int32*	Event number whose specific values are assigned by the Service.
Data	*int32*	Additional numeric data. Specific values vary by the *EventNumber* and the Service. This property is settable.
Obj	*Object*	Additional data whose usage varies by the *EventNumber* and Service. This property is settable.

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

See Also **"Events"** on page Intro-19, **directIO** Method.

StatusUpdateEvent

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

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

Attributes This event contains the following attribute:

Attribute	Type	Description
Status	*int32*	Reports a change in the power status of a light. ***Note that Release 1.3*** added Power State Reporting with additional *Power reporting* **StatusUpdateEvent** *values.* The Update Firmware capability, added in ***Release 1.9***, added additional *Status* values for communicating the status/progress of an asynchronous update firmware process. See "**StatusUpdateEvent**" description on page 1-34.

Remarks Enqueued when the light detects a power state change.

See Also **"Events"** on page Intro-19.

CHAPTER 29

POS Power

This Chapter defines the POS Power device category.

Summary

Properties (UML attributes)

Common	*Type*	*Mutability*	*Version*	*May Use After*
AutoDisable:	*boolean*	{read-write}	1.5	Not Supported
CapCompareFirmwareVersion:	*boolean*	{read-only}	1.9	open
CapPowerReporting:	*int32*	{read-only}	1.3	open
CapStatisticsReporting:	*boolean*	{read-only}	1.8	open
CapUpdateFirmware:	*boolean*	{read-only}	1.9	open
CapUpdateStatistics:	*boolean*	{read-only}	1.8	open
CheckHealthText:	*string*	{read-only}	1.5	open
Claimed:	*boolean*	{read-only	1.5	open
DataCount:	*int32*	{read-only}	1.5	Not Supported
DataEventEnabled:	*boolean*	{read-write}	1.5	Not Supported
DeviceEnabled:	*boolean*	{read-write}	1.5	open, claim
FreezeEvents:	*boolean*	{read-write}	1.5	open
OutputID:	*int32*	{read-only}	1.5	Not Supported
PowerNotify:	*int32*	{read-write}	1.5	open
PowerState:	*int32*	{read-only}	1.5	open
State:	*int32*	{read-only}	1.5	--
DeviceControlDescription:	*string*	{read-only}	1.5	--
DeviceControlVersion:	*int32*	{read-only}	1.5	--
DeviceServiceDescription:	*string*	{read-only}	1.5	open
DeviceServiceVersion:	*int32*	{read-only}	1.5	open
PhysicalDeviceDescription:	*string*	{read-only}	1.5	open
PhysicalDeviceName:	*string*	{read-only}	1.5	open

Properties (Continued)

Specific	*Type*	*Mutability*	*Version*	*May Use After*
BatteryCapacityRemaining:	*int32*	{read-only}	1.9	open
BatteryCriticallyLowThreshold:	*int32*	{read-write}	1.9	open
BatteryLowThreshold:	*int32*	{read-write}	1.9	open
CapBatteryCapacityRemaining:	*boolean*	{read-only}	1.9	open
CapChargeTime:	*boolean*	{read-only}	1.16	open
CapFanAlarm:	*boolean*	{read-only}	1.5	open
CapHeatAlarm:	*boolean*	{read-only}	1.5	open
CapQuickCharge:	*boolean*	{read-only}	1.5	open
CapRestartPOS:	*boolean*	{read-only}	1.9	open
CapShutdownPOS:	*boolean*	{read-only}	1.5	open
CapStandbyPOS:	*boolean*	{read-only}	1.9	open
CapSuspendPOS:	*boolean*	{read-only}	1.9	open
CapTimeMode:	*boolean*	{read-only}	1.16	open
CapUPSChargeState:	*int32*	{read-only}	1.5	open
CapVariableBatteryCriticallyLowThreshold:	*boolean*	{read-only}	1.9	open
CapVariableBatteryLowThreshold:	*boolean*	{read-only}	1.9	open
ChargeTime:	*int32*	{read-only}	1.16	open
EnforcedShutdownDelayTime:	*int32*	{read-write}	1.5	open
PowerFailDelayTime:	*int32*	{read-only}	1.5	open
PowerSource:	*int32*	{read-only}	1.9	open
QuickChargeMode:	*boolean*	{read-only}	1.5	open
QuickChargeTime:	*int32*	{read-only}	1.5	open
TimeMode:	*boolean*	{read-write}	1.16	open
UPSChargeState:	*int32*	{read-only}	1.5	open, claim & enable

Methods (UML operations)

Common

Name	*Version*
open (logicalDeviceName: *string*): **void {raises-exception}**	1.5
close ( ): void {raises-exception, use after open}	1.5
claim (timeout: *int32* **): void {raises-exception, use after open}**	1.5
release ( ): void {raises-exception, use after open, claim}	1.5
checkHealth (level: *int32*): **void {raises-exception, use after open, enable}**	1.5
clearInput ( ): void { }	***Not supported***
clearInputProperties ( ): void { }	***Not supported***
clearOutput ( ): void { }	***Not supported***
directIO (command: *int32,* **inout data:** *int32,* **inout obj:** *object*): **void {raises-exception, use after open}**	1.5
compareFirmwareVersion (firmwareFileName: *string,* **out result:** *int32*): **void {raises-exception, use after open, claim, enable}**	1.9
resetStatistics (statisticsBuffer: *string*): **void {raises-exception, use after open, claim, enable}**	1.8
retrieveStatistics (inout statisticsBuffer: *string*): **void {raises-exception, use after open, claim, enable}**	1.8
updateFirmware (firmwareFileName: *string*): **void {raises-exception, use after open, claim, enable}**	1.9
updateStatistics (statisticsBuffer: *string*): **void {raises-exception, use after open, claim, enable}**	1.8

Specific

Name	
restartPOS ( ): void {raises-exception, use after open, enable}	1.9
shutdownPOS ( ): void {raises-exception, use after open, enable}	1.5
standbyPOS (reason: int32): void {raises-exception, use after open, enable}	1.9
suspendPOS (reason: int32): void {raises-exception, use after open, enable}	1.9

Events (UML interfaces)

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

General Information

The POS Power programmatic name is “POSPower”.

Capabilities

The POSPower device class has the following capabilities:

- Supports a command to “shut down” the system.
- Supports a command to restart the system.
- Supports a command to “suspend” the system.
- Supports a command to have the system go to standby.
- Supports accessing a power handling mechanism of the underlying operating system and hardware.
- Informs the application if a power fail situation has occurred.
- Informs the application about battery level.
- Informs the application if the UPS charge state has changed.
- Informs the application about high CPU temperature.
- Informs the application about stopped CPU fan.
- Informs the application if an operating system dependent enforced shutdown mechanism is processed.
- Allows the application after saving application data locally or transferring application data to a server to shut down the POS terminal.
- Informs the application about an initiated shutdown.

Device Sharing

The POSPower is a sharable device. Its device sharing rules are:

- After opening and enabling the device, the application may access all properties and methods and will receive status update events.
- If more than one application has opened and enabled the device, all applications may access its properties and methods. Status update events are fired to all of the applications.
- If one application claims the POSPower, then only that application may call the **shutdownPOS, standbyPOS, or suspendPOS** methods. This feature provides a degree of security, such that these methods may effectively be restricted to the main POS application if that application claims the device at startup.
- See the “Summary” table for precise usage prerequisites.

Model

The general model of POSPower is based on the power model of each device in version 1.3 or later. The same common properties are used but all states relate to the POS terminal itself and not to a peripheral device.

There are three states of the POSPower:

- ONLINE. The POS terminal is powered on and ready for use. This is the “operational” state.
- OFF. The POS terminal is powered off or detached from the power supplying net. The POS terminal runs on battery power support. This is the powerfail situation.
- OFFLINE. The POS terminal is powered on but is running in a “lower-power-consumption” mode. It may need to be placed online by pressing a button or key or something else which may wake up the system.

Power reporting only occurs while the device is open, enabled and power notification is switched on.

In a powerfail situation - that means the POSPower is in the state OFF - the POS terminal will be shut down automatically after the last application has closed the POSPower device or the time specified by the **EnforcedShutdownDelayTime** property has been elapsed.

A call to the **shutdownPOS** method will always shut down the POS terminal independent of the system power state.

Version 1.9 or later

Support of battery powered devices is added. In addition to adding properties to report battery levels and power sources, properties are added to allow for the setting of low and critically low battery levels. The POSPower device also includes the ability to request or respond to request to enter the standby and suspend states. The model does not attempt to duplicate other power management models such as APM and ACPI, but leaves those implementation details to the provider. As a rule, the suspend state will consume less power than the standby state, which in turn will consume less power than the on state. A suggested mapping of these states to other power management models is:

State	*ACPI*	*APM*	*Description*
On	S0	ON	Active, Powered On
Standby	S1	SUSPEND	Displays and drives off, CPU, RAM and fans powered on
Suspend	S3	SUSPEND	Only RAM powered
Off	S5	OFF	Completely powered off

POSPower Class Diagram

Updated in Release 1.16

The following diagram shows the relationships between the POSPower classes.

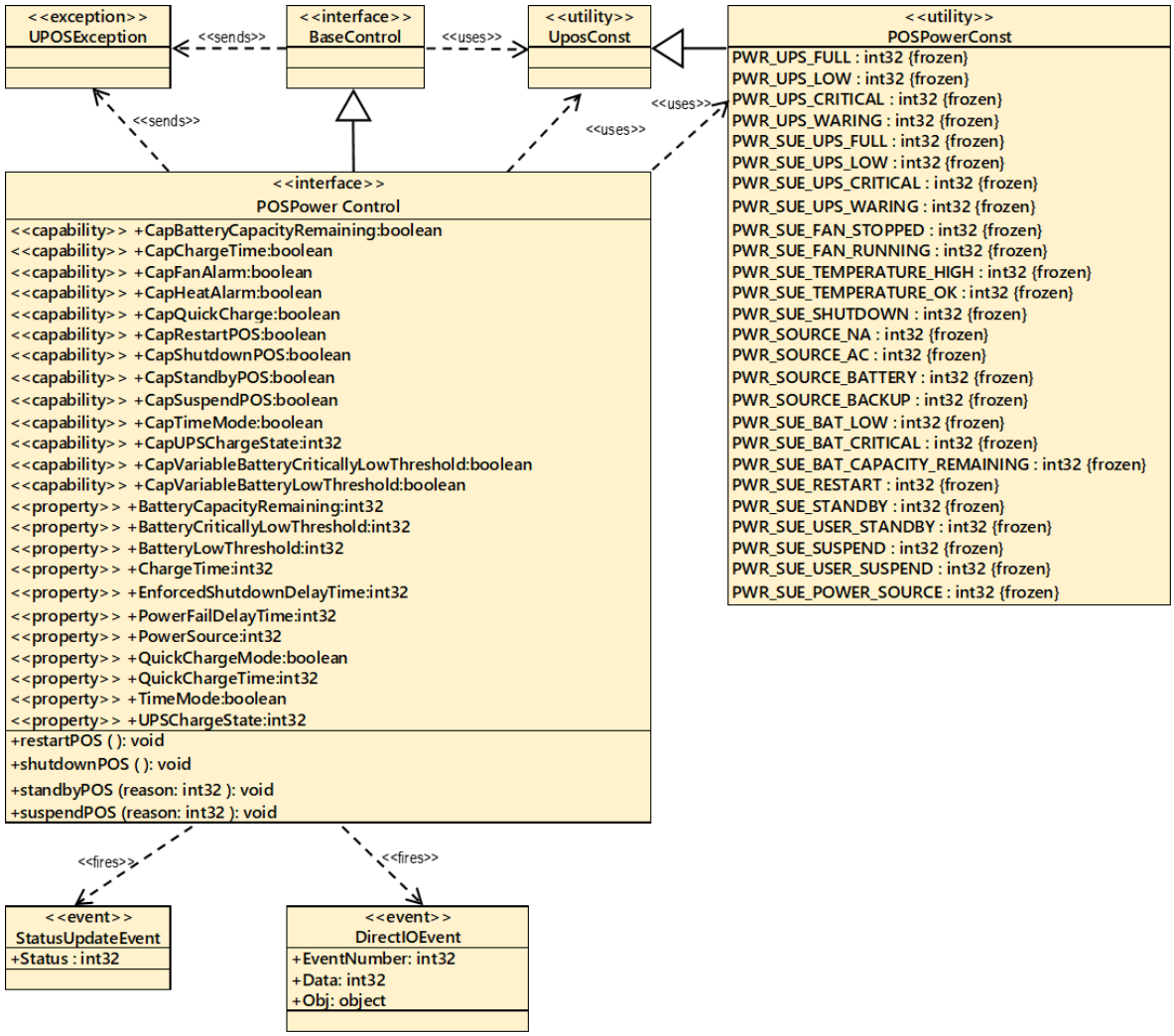


Fig. Chap.29-1 POSPower Class Diagram

POSPower Sequence Diagram

The following sequence diagram shows the typical usage of the POSPower device for registering for **StatusUpdateEvent**s and an atypical case of initiating a **shutdownPOS** call.

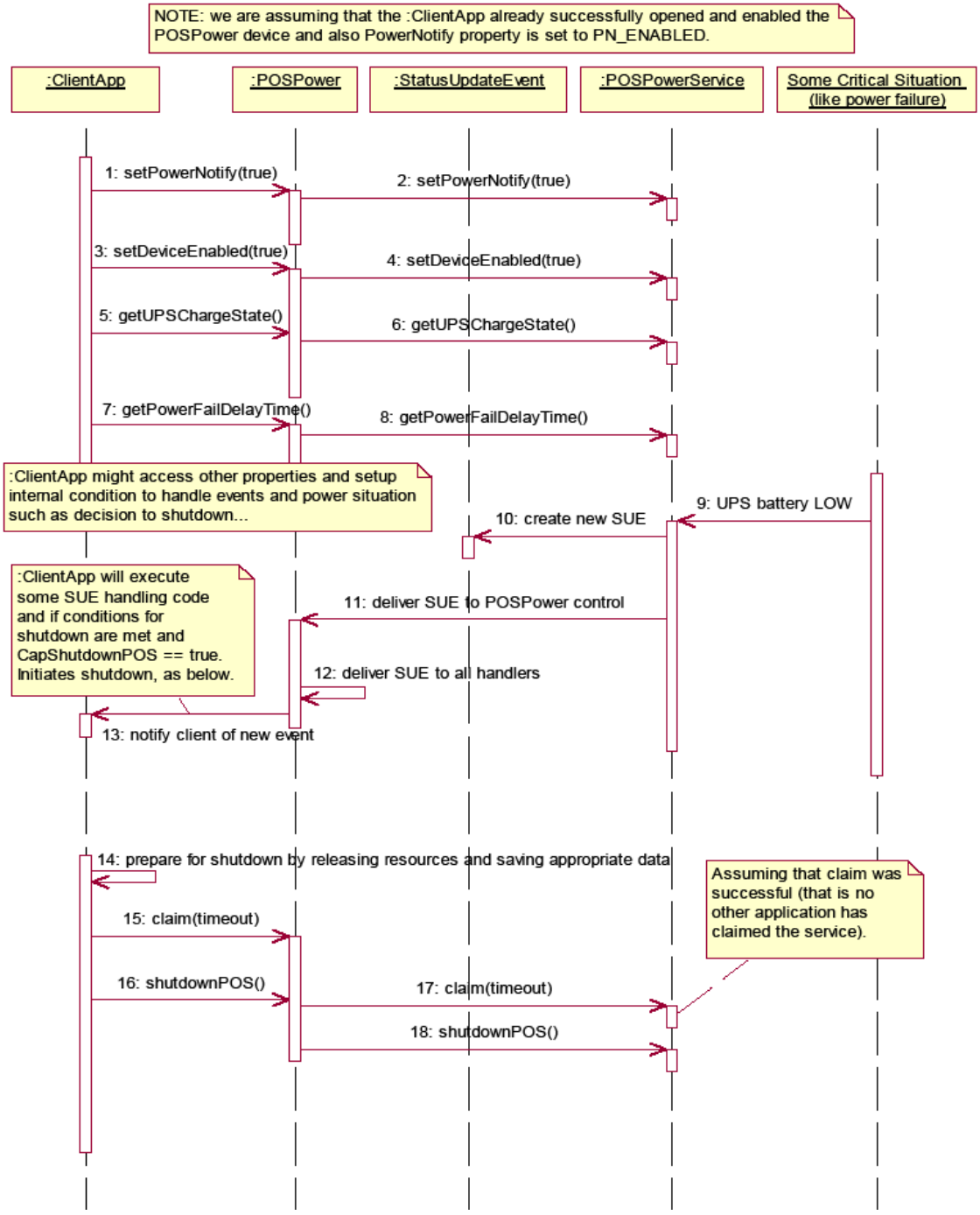


Fig. Chap. 29-2 POSPower Sequence Diagram

POSPower Standby Sequence Diagram

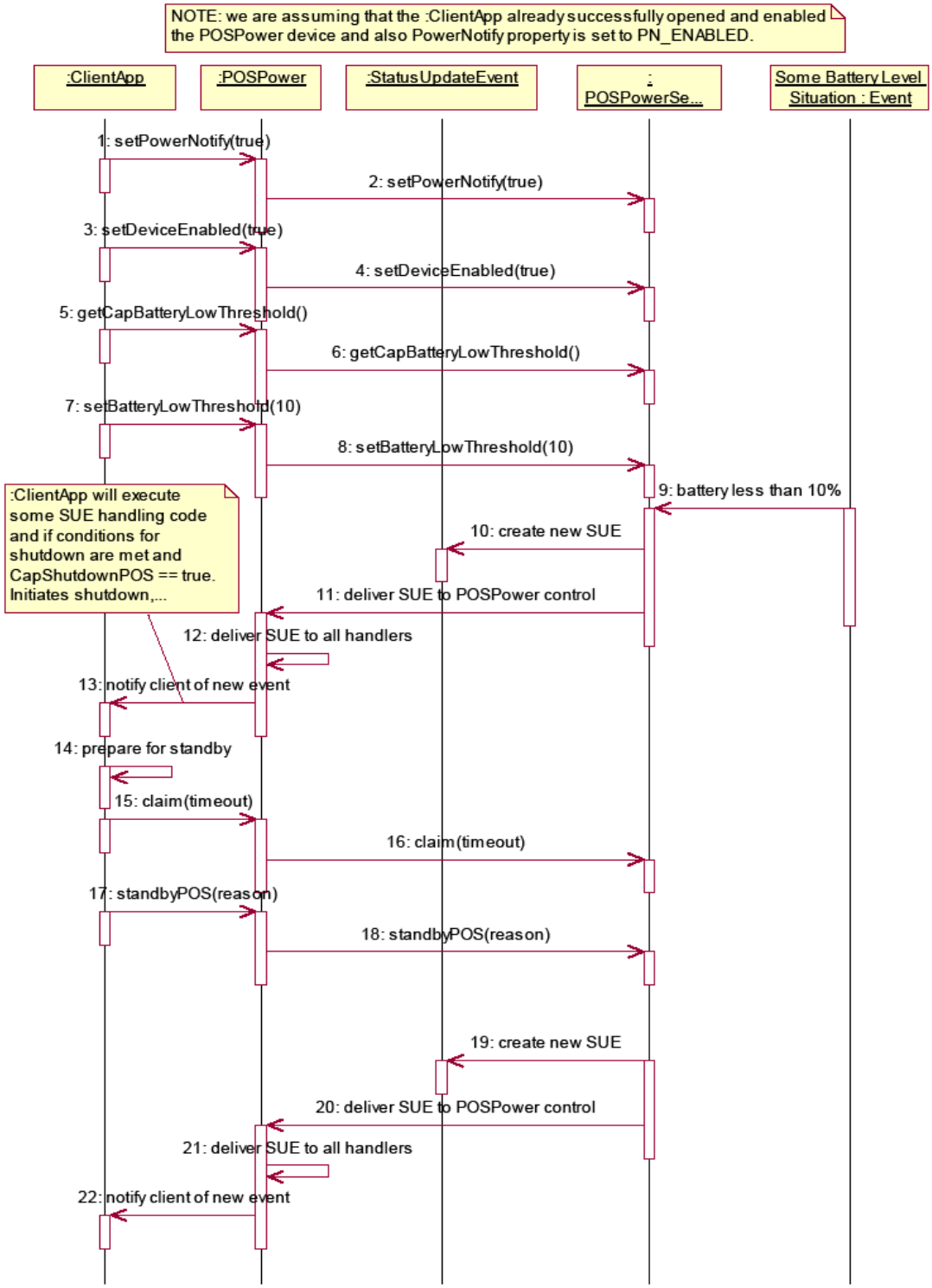


Fig. Chap. 29-3 POSPower Standby Sequence Diagram

POSPower State Diagram

The following state diagram depicts the POSPower Control device model.

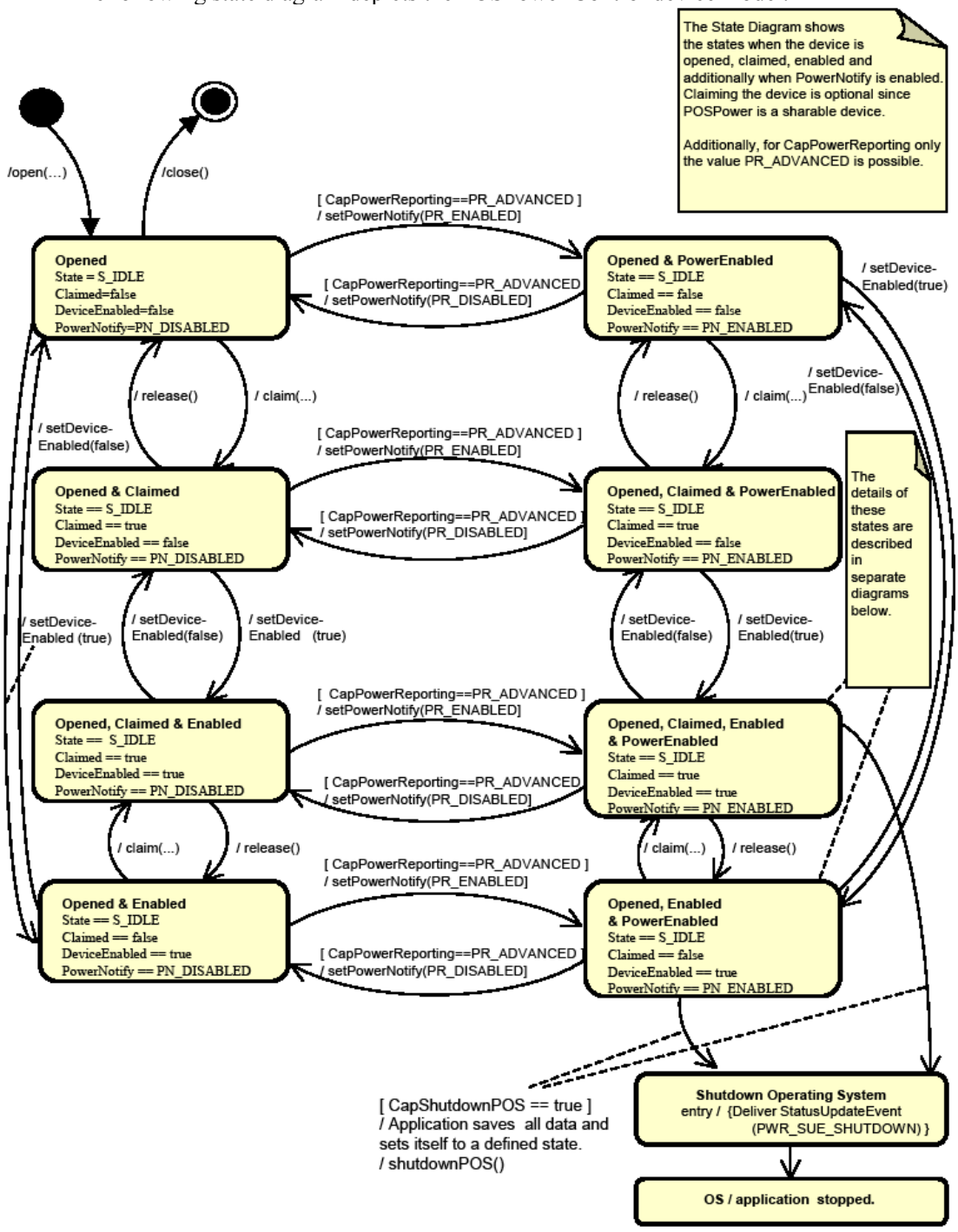


Fig. Chap. 29-4 Power State Diagram (POSPoer Control Device Model)**POSPower PowerState Diagram - Part 1**

The following state diagram depicts the POSPower Power States.

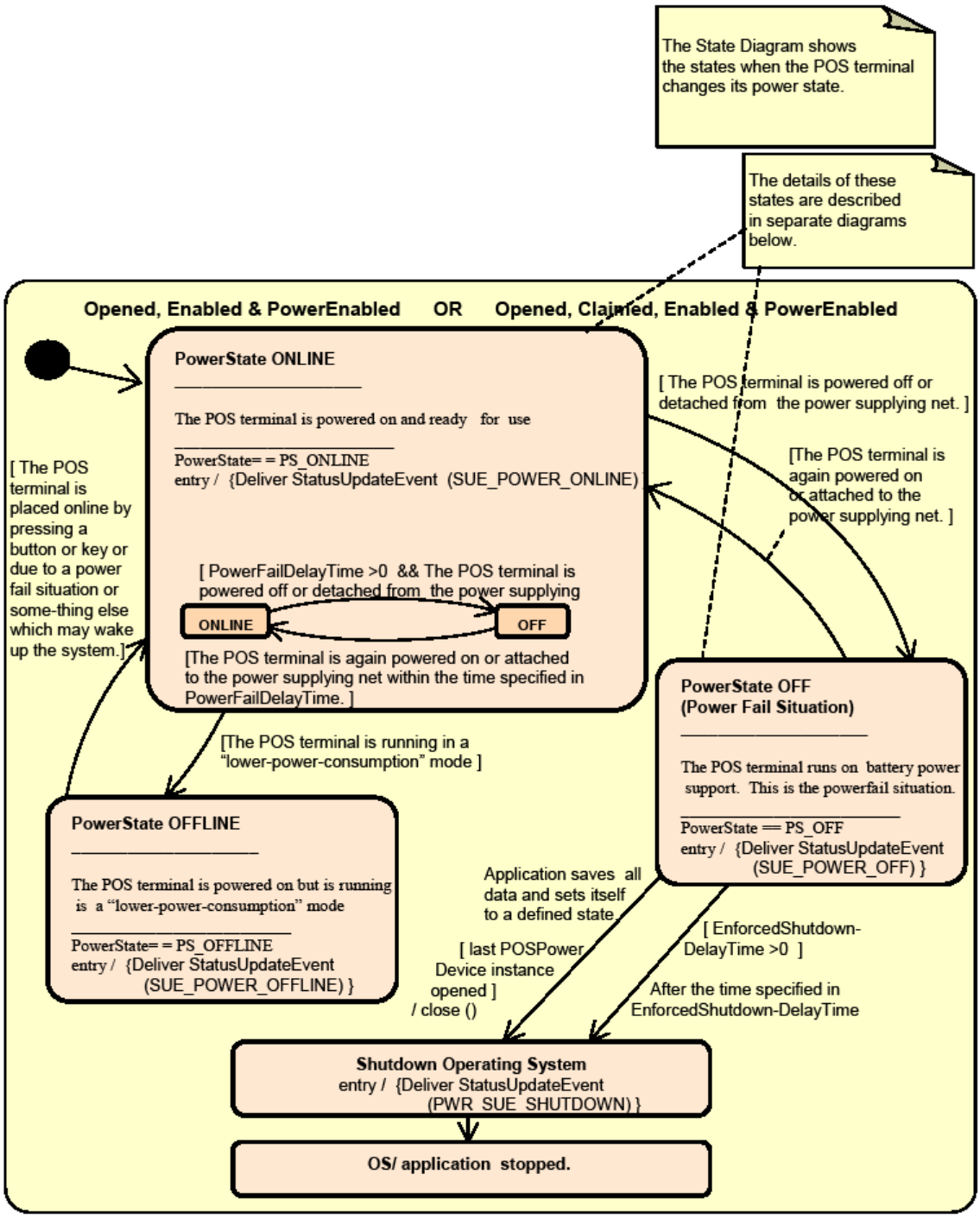


Fig. Chap. 29-5 POSPower PowerState Diagram (Part 1)

POSPower PowerState Diagram - Part 2

The following state diagram depicts the POSPower PowerState ONLINE.

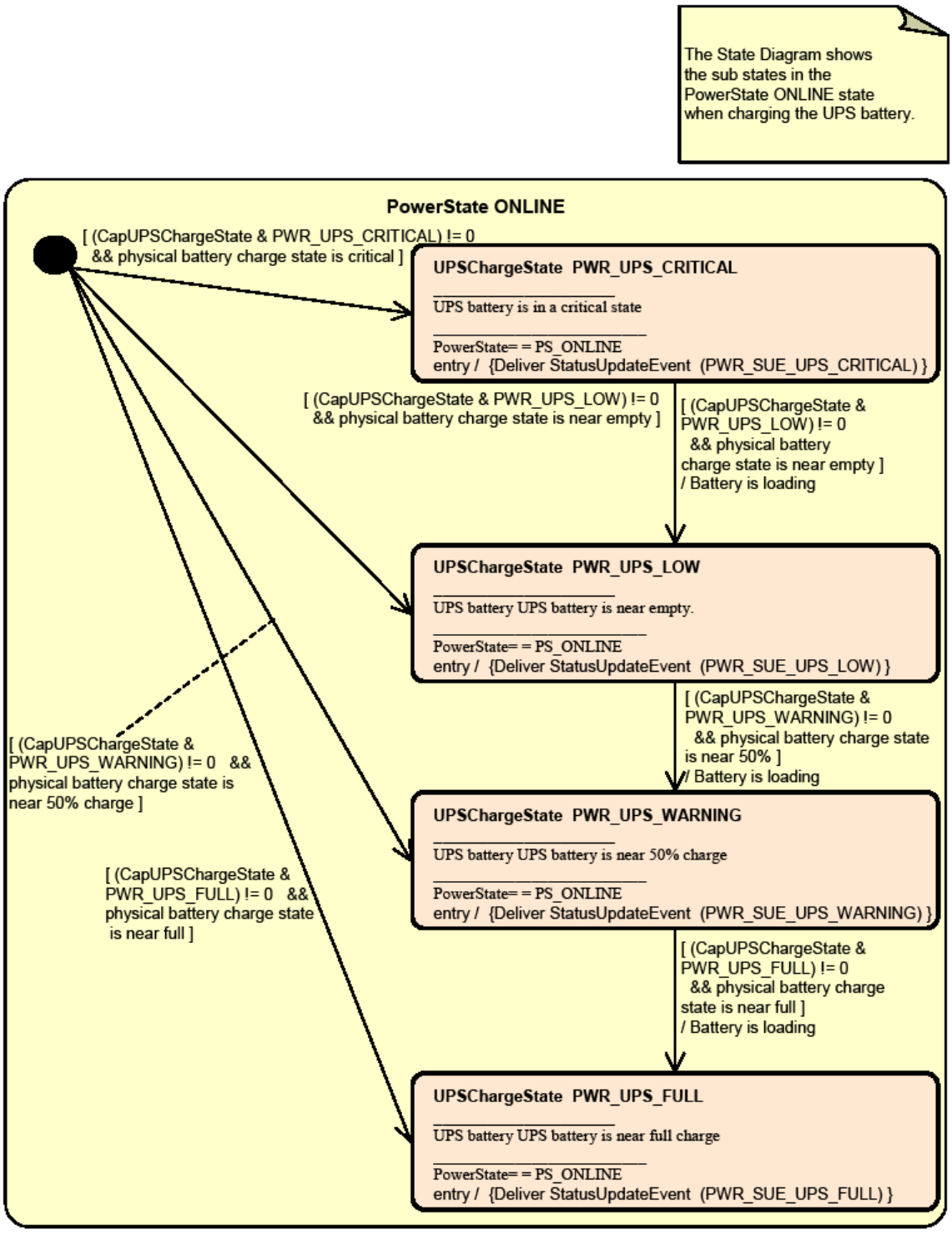


Fig. Chap. 29-6 POSPower PowerState Diagram (Part 2)

POSPower PowerState Diagram - Part 3

The following state diagram depicts the POSPower PowerState OFF.

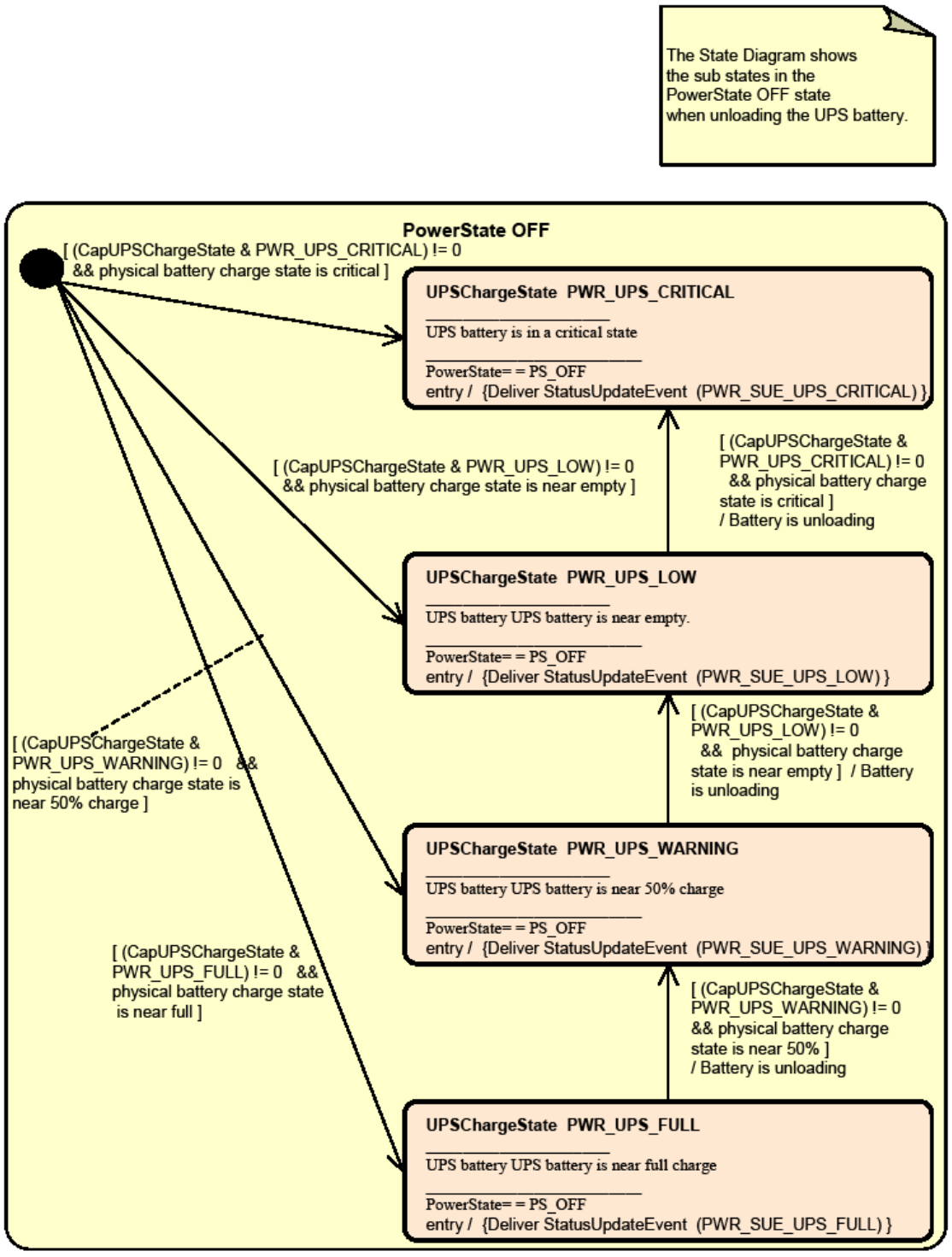


Fig. Chap. 29-7 POSPower PowerState Diagram (Part 3)

POSPower State Chart Diagram for Fan and Temperature

The following state diagram depicts the handling of fan and temperature alarms.

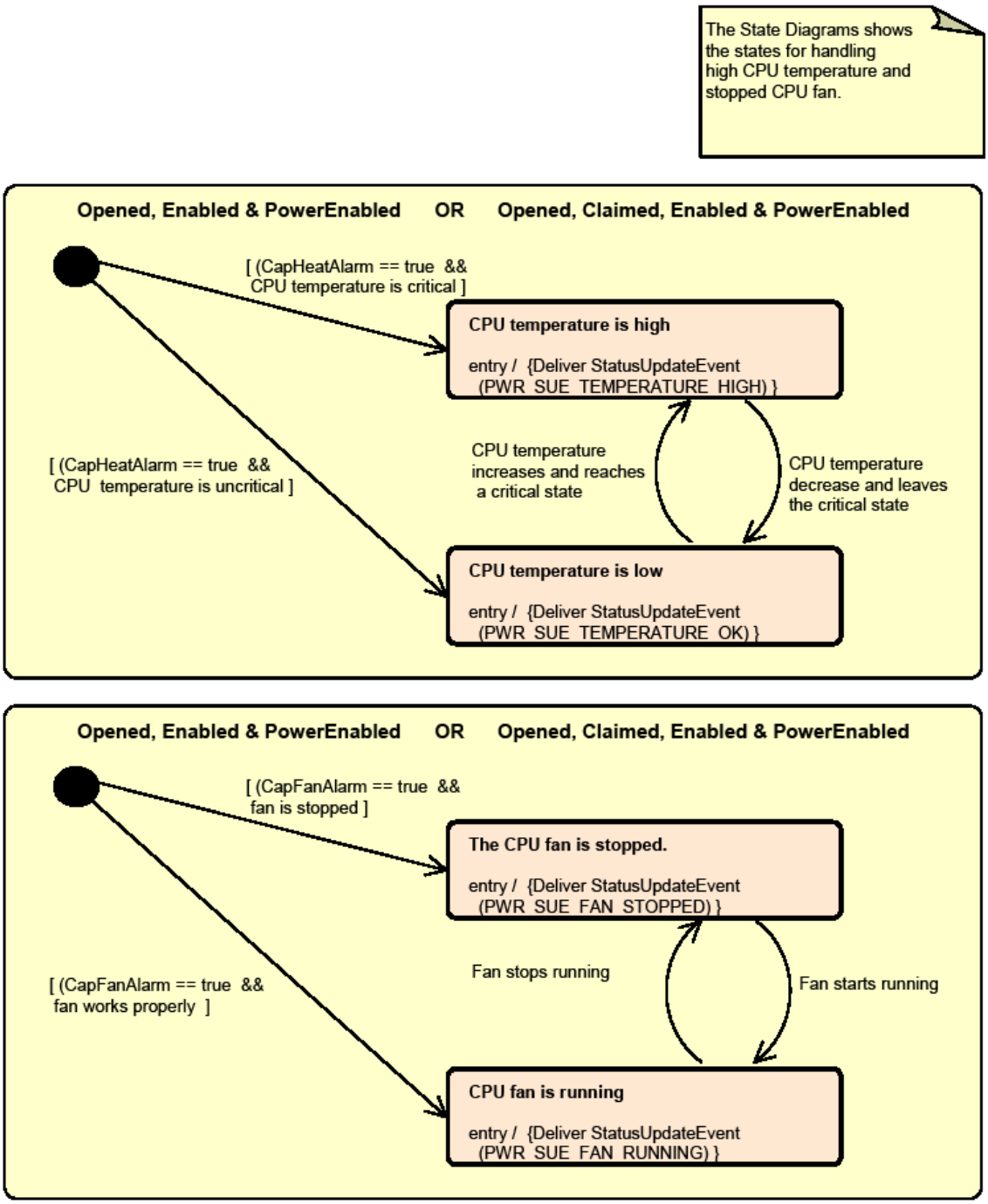


Fig. Chap. 29-8 POSPower State Chart Diagram (Fan and Temperature)

POSPower Battery State Diagram

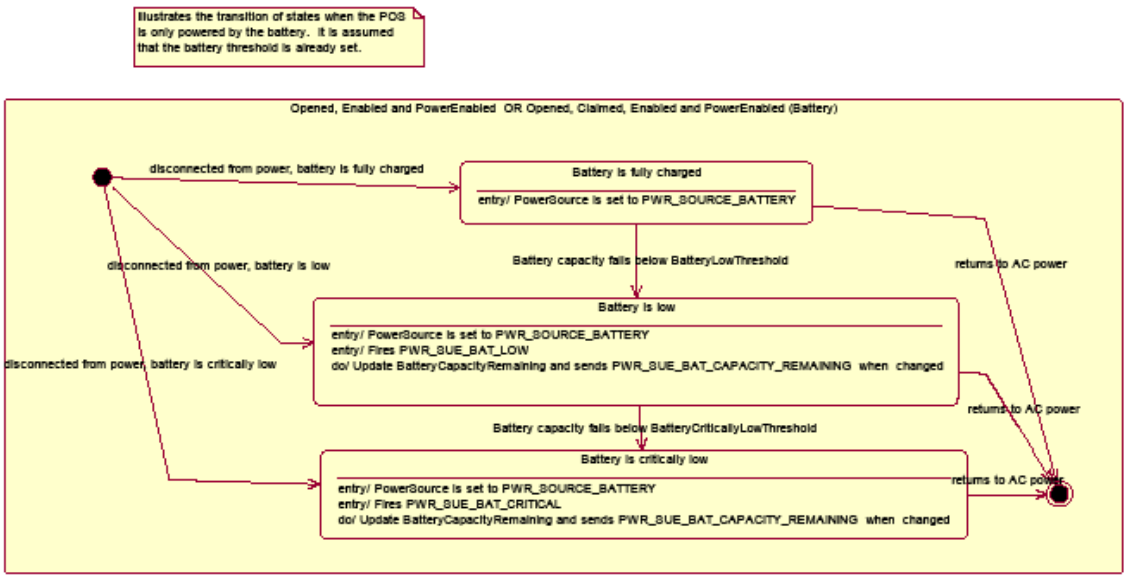


Fig. Chap. 29-9 POSPower Battery State Diagram

POSPower Power Transitions State Diagram

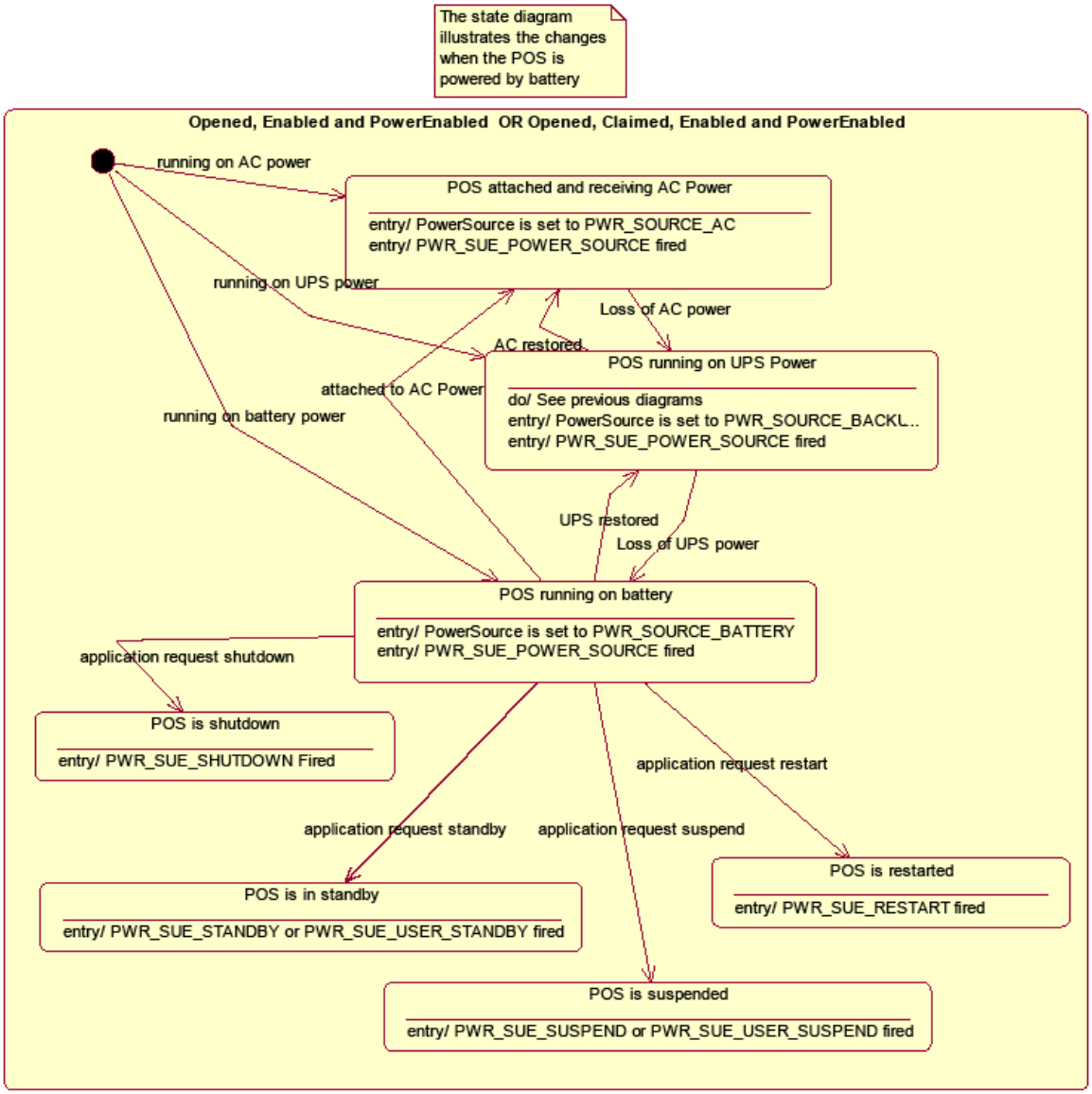


Fig. Chap. 29-10 POSPower Power Transitions State Diagram

Properties (UML attributes)

BatteryCapacityRemaining Property

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

Remarks A value of 0 to 100 represents percent of battery capacity remaining.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **CapBatteryCapacityRemaining** Property

BatteryCriticallyLowThreshold Property

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

Remarks If not zero, this property holds the threshold at which a **PWR_SUE_BAT_CRITICAL** Status Update Event is generated. The values 1 through 99 represent the percentage of the capacity remaining. The value 0 indicates that Battery Critically Low reporting is not supported or is disabled.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **CapVariableBatteryCriticallyLowThreshold** Property**, StatusUpdateEvent**

BatteryLowThreshold Property

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

Remarks If not zero, this property holds the threshold at which a **PWR_SUE_BAT_LOW** Status Update Event is generated. The value 1 to 99 represents the percent capacity remaining. The value 0 indicates that battery low reporting is not supported or is disabled. If variable battery low threshold is supported, setting a value between 1 and 99 sets the threshold to that value. Setting a value of zero disables battery low reporting.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **CapVariableBatteryLowThreshold** Property**, StatusUpdateEvent**

CapBatteryCapacityRemaining Property

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

Remarks If true, the device is able to provide battery capacity information. Otherwise it is false.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **BatteryCapacityRemaining** Property

CapChargeTime Property *Added in Release 1.16*

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

Remarks If true, the device is able to acquire the remaining time until full charging. Otherwise it is false.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **ChargeTime** Property.

CapFanAlarm Property

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

Remarks If true, the device is able to detect whether the CPU fan is stopped. Otherwise it is false.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

CapHeatAlarm Property

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

Remarks If true the device is able to detect whether the CPU is running at too high of a temperature. Otherwise it is false.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

CapQuickCharge Property

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

Remarks If true, the power management allows the charging of the UPS battery in quick mode. The time for charging the battery is shorter than usual. Otherwise it is false.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **QuickChargeMode** Property, **QuickChargeTime** Property.

CapRestartPOS Property

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

Remarks If true the device is able to explicitly restart the POS. Otherwise it is false.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **restartPOS** Method.

CapShutdownPOS Property

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

Remarks If true the device is able to explicitly shut down the POS. Otherwise it is false.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **shutdownPOS** Method.

CapStandbyPOS Property

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

Remarks If true, the device is able to request that the POS System enter the Standby state. Otherwise it is false.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **standbyPOS** Method.

CapSuspendPOS Property

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

Remarks If true, the device is able to request that the POS System enter the Suspend state. Otherwise it is false.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **suspendPOS** Method.

CapTimeMode Property *Added in Release 1.16*

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

Remarks If true the device is able to switch the unit of battery remaining / threshold related property value to seconds. Otherwise it is false. This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **TimeMode** Property

CapUPSChargeState Property

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

Remarks If not equal to zero, the UPS can deliver one or more charge states. It can contain any of the following values logically ORed together.

Value	Meaning
PWR_UPS_FULL	UPS battery is near full charge.
PWR_UPS_WARNING	UPS battery is near 50% charge.
PWR_UPS_LOW	UPS battery is near empty. Application shutdown should be started to ensure that is can be completed before the battery charge is depleted. A minimum of 2 minutes of normal system operation can be assumed when this state is entered unless this is the first state reported upon entering the "Off" power state.
PWR_UPS_CRITICAL	UPS battery is in a critical state and could be disconnected at any time without further warning. This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **UPSChargeState** Property.

CapVariableBatteryCriticallyLowThreshold Property

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

Remarks If true, the device supports a variable threshold for critically low battery. Otherwise it is false.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **BatteryCriticallyLowThreshold** Property**, StatusUpdateEvent**

CapVariableBatteryLowThreshold Property

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

Remarks If true, the device supports a variable threshold for battery low. Otherwise it is false.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **BatteryLowThreshold** Property**, StatusUpdateEvent**

ChargeTime Property *Added in Release 1.16*

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

Remarks Indicates the time remaining until the battery is fully charged in seconds.

If equal to zero the battery is not charging or not supported.

This property is only set if **CapChargeTime** is true.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **CapChargeTime** Property.

EnforcedShutdownDelayTime Property

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

Remarks If not equal to zero the system has a built-in mechanism to shut down the POS terminal after a determined time in a power fail situation. This property contains the time in milliseconds when the system will shut down automatically after a power failure. A power failure is the situation when the POS terminal is powered off or detached from the power supplying net and runs on UPS.
If zero no automatic shutdown is performed and the application has to call itself the **shutdownPOS** method.
Applications will be informed about an initiated automatic shutdown**.**
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **shutdownPOS** Method.

PowerFailDelayTime Property

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

Remarks This property contains the time in milliseconds for power fail intervals which will not create a power fail situation. In some countries the power has sometimes short intervals where the power supply is interrupted. Those short intervals are in the range of milliseconds up to a few seconds and are handled by batteries or other electric equipment and should not cause a power fail situation. The power fail interval starts when the POS terminal is powered off or detached from the power supplying net and runs on UPS. The power fail interval ends when the POS terminal is again powered on or attached to the power supplying net. However, if the power fail interval is longer than the time specified in the **PowerFailDelayTime** property a power fail situation is created.

Usually this parameter is a configuration parameter of the underlying power management. So, the application can only read this property.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

PowerSource Property

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

Remarks This property holds the current power source if power source reporting is available. A StatusUpdateEvent is generated each time this property is updated.

Value	Meaning
PWR_SOURCE_NA	Power source reporting is not available.
PWR_SOURCE_AC	The current power source is the AC line.
PWR_SOURCE_BATTERY	The current power source is a system battery. This value is only presented for systems that operate normally on battery.
PWR_SOURCE_BACKUP	The current power source is a backup source such as an UPS or backup battery.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **StatusUpdateEvent**

QuickChargeMode Property

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

Remarks If true, the UPS battery is being recharged in a quick charge mode.
If false, it is being charged in a normal mode.

This property is only set if **CapQuickCharge** is true.

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **CapQuickCharge** Property, **QuickChargeTime** Property.

QuickChargeTime Property

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

Remarks This time specifies the remaining time for charging the UPS battery in quick charge mode. After the time has elapsed, the UPS battery charging mechanism of power management usually switches into normal mode.

This time is specified in milliseconds.

This property is only set if **CapQuickCharge** is true.

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **CapQuickCharge** Property, **QuickChargeTime** Property.

TimeMode Property *Added in Release 1.16*

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

Remarks If true, the value of the battery remaining / threshold related property is in seconds.

If false, the value of the battery remaining / threshold related property is in percent.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20

See Also **CapTimeMode** Property, **BatteryCapacityRemaining** Property, **BatteryCriticallyLowThreshold** Property, **BatteryLowThreshold** Property.

UPSChargeState Property

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

Remarks This property holds the actual UPS charge state.

It has one of the following values:

Value	Meaning
PWR_UPS_FULL	UPS battery is near full charge.
PWR_UPS_WARNING	UPS battery is near 50% charge.
PWR_UPS_LOW	UPS battery is near empty. Application shutdown should be started to ensure that is can be completed before the battery charge is depleted. A minimum of 2 minutes of normal system operation can be assumed when this state is entered unless this is the first state reported upon entering the "Off" power state.
PWR_UPS_CRITICAL	UPS battery is in a critical state and could be disconnected at any time without further warning.

This property is initialized and kept current while the device is enabled.

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20

See Also **CapUPSChargeState** Property.

Methods (UML operations)

restartPOS Method

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

Remarks Call to restart the POS terminal. This method will always restart the system independent of the system power state.

If the POSPower is claimed, only the application which claimed the device is able to restart the POS terminal.

Applications will be informed about an initiated restart.

Errors A UposException may be thrown when this method is invoked. For further information, see **"Errors"** on page Intro-20

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

Value	Meaning
E_ILLEGAL	This method is not supported (see the **CapRestartPOS** property**)**

See Also **CapRestartPOS** Property

shutdownPOS Method

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

Remarks Call to shut down the POS terminal. This method will always shut down the system independent of the system power state.

If the POSPower is claimed, only the application which claimed the device is able to shut down the POS terminal.

Applications will be informed about an initiated shutdown.

It is recommended that in a power fail situation an application has to call this method after saving all data and setting the application to a defined state.
If the **EnforcedShutdownDelayTime** property specifies a time greater than zero and the application did not call the **shutdownPOS** method within the time specified in **EnforcedShutdownDelayTime**, the system will be shut down automatically. This mechanism may be provided by an underlying operating system to prevent the battery from being emptied before the system is shut down. This method is only supported if **CapShutdownPOS** is true.

Errors A UposException may be thrown when this method is invoked. For further information, see **"Errors"** on page Intro-20

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

Value	Meaning
E_ILLEGAL	This method is not supported. (See the **CapshutdownPOS** property**)**

See Also **CapShutdownPOS** Property, **EnforcedShutdownDelayTime** Property.

standbyPOS Method

Syntax **standbyPOS (reason: *int32*):**
void {raises-exception, use after open, enable}

Remarks Call to request that the system be placed into the Standby state or to respond to a request from the system, OS or other application that the system be put into Standby state.

The *reason* parameter indicates the reason the POS terminal should enter a standby state:

Value	Description
PWR_REASON_REQUEST	Call is to request that the system enter the standby state.
PWR_REASON_ALLOW	Call is a response to a standby Status Update Event and specifies that the request should be allowed.
PWR_REASON_DENY	Call is a response to a standby Status Update Event and specifies that the request should be denied.

Errors A UposException may be thrown when this method is invoked. For further information, see **"Errors"** on page Intro-20

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

Value	Meaning
E_ILLEGAL	This method is not supported (see the **CapStandbyPOS** property**)**

See Also **CapStandbyPOS** Property.

suspendPOS Method

Syntax **suspendPOS (reason: *int32*):**
void {raises-exception, use after open, enable}

Remarks Call to request that the system be placed into the Suspend state or to respond to a request from the system, OS or other application that the system be put into Suspend state.

The *reason* parameter indicates the reason the POS terminal should enter a standby state:

Value	Description
PWR_REASON_REQUEST	Call is to request that the system enter the suspend state.
PWR_REASON_ALLOW	Call is a response to a suspend Status Update Event and specifies that the request should be allowed.
PWR_REASON_DENY	Call is a response to a suspend Status Update Event and specifies that the request should be denied.

Errors A UposException may be thrown when this method is invoked. For further information, see **"Errors"** on page Intro-20

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

Value	Meaning
E_ILLEGAL	This method is not supported (see the **CapSuspendPOS** property**)**

See Also **CapSuspendPOS** Property.

Events (UML Interfaces)

DirectIOEvent

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

EventNumber : *int32* {read-only}
Data : *int32* {read-write}
Obj : *object*{read-write}

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

Attributes This event contains the following attributes:

Attributes	Type	Description
EventNumber	*int32*	Event number whose specific values are assigned by the Service.
Data	*int32*	Additional numeric data. Specific values vary by the *EventNumber* and the Service. This property is settable.
Obj	*object*	Additional data whose usage varies by the E*ventNumber* and Service. This property is settable.

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

See Also **"Errors"** on page Intro-20, **directIO** Method.

StatusUpdateEvent

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

Description Delivered when **UPSChargeState** changes or an alarm situation occurs.

Attributes This event contains the following attribute:

Attributes	Type	Description
Status	*int32*	See below.

The *Status* property contains the updated power status or alarm status.

Value	Meaning
PWR_SUE_UPS_FULL	UPS battery is near full charge. Can be returned if **CapUPSChargeState** contains PWR_UPS_FULL.
PWR_SUE_UPS_WARNING	UPS battery is near 50% charge. Can be returned if CapUPSChargeState contains PWR_UPS_WARNING.
PWR_SUE_UPS_LOW	UPS battery is near empty. Application shutdown should be started to ensure that it can be completed before the battery charge is depleted. A minimum of 2 minutes of normal system operation can be assumed when this state is entered unless this is the first charge state reported upon entering the "Off" state. Can be returned if **CapUPSChargeState** contains PWR_UPS_LOW.
PWR_SUE_UPS_CRITICAL	UPS is in critical state, and will in short time be disconnected. Can be returned if **CapUPSChargeState** contains PWR_UPS_CRITICAL.
PWR_SUE_FAN_STOPPED	The CPU fan is stopped. Can be returned if **CapFanAlarm** is true.
PWR_SUE_FAN_RUNNING	The CPU fan is running. Can be returned if **CapFanAlarm** is true.

PWR_SUE_TEMPERATURE_HIGH

The CPU is running on high temperature. Can be returned if **CapHeatAlarm** is true.

PWR_SUE_TEMPERATURE_OK

The CPU is running on normal temperature. Can be returned if **CapHeatAlarm** is true.

PWR_SUE_SHUTDOWN

The system will shut down immediately.

PWR_SUE_BAT_LOW

The system remaining battery capacity is at or below the low battery threshold and the system is operating from the battery.

PWR_SUE_BAT_CRITICAL

The system remaining battery capacity is at or below the critically low battery threshold and the system is operating from the battery.

PWR_SUE_BAT_CAPACITY_REMAINING.

The **BatteryCapacityRemaining** property has been updated

PWR_SUE_RESTART

The system will restart immediately.

PWR_SUE_STANDBY

The system is requesting a transition to the **Standby** state

PWR_SUE_USER_STANDBY

The system is requesting a transition to the **Standby** state as a result of user input.

PWR_SUE_SUSPEND

The system is requesting a transition to the **Suspend** state.

PWR_SUE_USER_SUSPEND

The system is requesting a transition to the **Suspend** state as a result of user input.

PWR_SUE_PWR_SOURCE

The **PowerSource** property has been updated.

Note** that **Release 1.3 added Power State Reporting with additional *Power reporting* **StatusUpdateEvent** *values.*

The Update Firmware capability, added in ***Release 1.9***, added additional *Status* values for communicating the status/progress of an asynchronous update firmware process. See "**StatusUpdateEvent**" description on page 1-34.

See Also **CapFanAlarm** Property, **CapHeatAlarm** Property, **CapUPSChargeState** Property**, UPSChargeState** Property.

CHAPTER 39

Video Capture

This Chapter defines the Video Capture device category.

Summary

Properties (UML attributes)

Common	*Type*	*Mutability*	*Version*	*May Use After*
AutoDisable:	*boolean*	{read-write}	1.16	open
CapCompareFirmwareVersion:	*boolean*	{read-only}	1.16	open
CapPowerReporting:	*int32*	{read-only}	1.16	open
CapStatisticsReporting:	*boolean*	{read-only}	1.16	open
CapUpdateFirmware:	*boolean*	{read-only}	1.16	open
CapUpdateStatistics:	*boolean*	{read-only}	1.16	open
CheckHealthText:	*string*	{read-only}	1.16	open
Claimed:	*boolean*	{read-only}	1.16	open
DataCount:	*int32*	{read-only}	1.16	open
DataEventEnabled:	*boolean*	{read-write}	1.16	open
DeviceEnabled:	*boolean*	{read-write}	1.16	open, claim
FreezeEvents:	*boolean*	{read-write}	1.16	open
OutputID:	*int32*	{read-only}	1.16	Not Supported
PowerNotify:	*int32*	{read-write}	1.16	open
PowerState:	*int32*	{read-only}	1.16	open
State:	*int32*	{read-only}	1.16	--
DeviceControlDescription:	*string*	{read-only}	1.16	--
DeviceControlVersion:	*int32*	{read-only}	1.16	--
DeviceServiceDescription:	*string*	{read-only}	1.16	open
DeviceServiceVersion:	*int32*	{read-only}	1.16	open
PhysicalDeviceDescription:	*string*	{read-only}	1.16	open
PhysicalDeviceName:	*string*	{read-only}	1.16	open

Properties (Continued)

Specific	*Type*	*Mutability*	*Version*	*May Use After*
CapCameraAutoExposition:	*boolean*	{read-only}	1.16	open
CapCameraAutoFocus:	*boolean*	{read-only}	1.16	open
CapCameraAutoGain:	*boolean*	{read-only}	1.16	open
CapCameraAutoWhiteBalance:	*boolean*	{read-only}	1.16	open
CapCameraBrightness:	*boolean*	{read-only}	1.16	open
CapCameraContrast:	*boolean*	{read-only}	1.16	open
CapCameraExposure:	*boolean*	{read-only}	1.16	open
CapCameraGain:	*boolean*	{read-only}	1.16	open
CapCameraHorizontalFlip:	*boolean*	{read-only}	1.16	open
CapCameraHue:	*boolean*	{read-only}	1.16	open
CapCameraSaturation:	*boolean*	{read-only}	1.16	open
CapCameraVerticalFlip:	*boolean*	{read-only}	1.16	open
CapCapture:	*boolean*	{read-only}	1.16	open
CapCaptureColorSpace:	*boolean*	{read-only}	1.16	open
CapCaptureColorSpaceList:	*string*	{read-only}	1.16	open
CapCaptureFrameRate:	*boolean*	{read-only}	1.16	open
CapCaptureMaxFrameRate:	*int32*	{read-only}	1.16	open
CapCaptureResolution:	*boolean*	{read-only}	1.16	open
CapCaptureResolutionList:	*string*	{read-only}	1.16	open
CapDecodeData:	*boolean*	{read-only}	1.16	open
CapIndividualRecognition:	*boolean*	{read-only}	1.16	open
CapPhotograph:	*boolean*	{read-only}	1.16	open
CapPhotographResolution:	*boolean*	{read-only}	1.16	open
CapPhotographResolutionList	*string*	{read-only}	1.16	open
CapPhotographType:	*boolean*	{read-only}	1.16	open
CapPhotographTypeList:	*string*	{read-only}	1.16	open
CapVideoRecording:	*boolean*	{read-only}	1.16	open

CapVideoRecordingFrameRate:	*boolean*	{read-only}	1.16	open
CapVideoRecordingMaxFrameRate:	*int32*	{read-only}	1.16	open
CapVideoRecordingResolution:	*boolean*	{read-only}	1.16	open
CapVideoRecordingResolutionList:	*string*	{read-only}	1.16	open
CapVideoRecordingType:	*boolean*	{read-only}	1.16	open
CapVideoRecordingTypeList:	*string*	{read-only}	1.16	open
BarCodeEnabled:	*boolean*	{read-write}	1.16	open, claim & enable
CameraAutoExposure:	*boolean*	{read-write}	1.16	open, claim & enable
CameraAutoFocus:	*boolean*	{read-write}	1.16	open, claim & enable
CameraAutoGain:	*boolean*	{read-write}	1.16	open, claim & enable
CameraAutoWhiteBalance:	*boolean*	{read-write}	1.16	open, claim & enable
CameraBrightness:	*int32*	{read-write}	1.16	open, claim & enable
CameraContrast:	*int32*	{read-write}	1.16	open, claim & enable
CameraExposure	*int32*	{read-write}	1.16	open, claim & enable
CameraGain:	*int32*	{read-write}	1.16	open, claim & enable
CameraHorizontalFlip:	*boolean*	{read-write}	1.16	open, claim & enable
CameraHue:	*int32*	{read-write}	1.16	open, claim & enable
CameraSaturation:	*int32*	{read-write}	1.16	open, claim & enable
CameraVerticalFlip:	*boolean*	{read-write}	1.16	open, claim & enable
CaptureColorSpace:	*string*	{read-write}	1.16	open, claim & enable
CaptureFrameRate:	*int32*	{read-write}	1.16	open, claim & enable
CaptureResolution:	*string*	{read-write}	1.16	open, claim & enable
IndividualRecognitionEnabled:	*boolean*	{read-write}	1.16	open, claim & enable
PhotographResolution:	*string*	{read-write}	1.16	open, claim & enable
PhotographType:	*int32*	{read-write}	1.16	open, claim & enable
VideoCaptureMode:	*int32*	{read-write}	1.16	open, claim & enable
VideoRecordingFrameRate:	*int32*	{read-write}	1.16	open, claim & enable
VideoRecordingResolution:	*int32*	{read-write}	1.16	open, claim & enable
VideoRecordingType:	*string*	{read-write}	1.16	open, claim & enable

Methods (UML operations)

Common

Name	*Version*
open (logicalDeviceName: *string*`)`**:** **void {raises-exception}**	1.16
close ( ): **void {raises-exception, use after open}**	1.16
claim (timeout: *int32*`)`**:** **void {raises-exception, use after open}**	1.16
release ( ): **void {raises-exception, use after open, claim}**	1.16
checkHealth (level: *int32*`)`**:** **void {raises-exception, use after open, enable}**	1.16
clearInput ( ): **void { }**	Not supported
clearInputProperties ( ): **void { }**	Not supported
clearOutput ( ): **void { }**	Not supported
directIO (command: *int32*`,` **inout data:** *int32*`,` **inout obj:** *object*`)`**:** **void {raises-exception, use after open}**	1.16
compareFirmwareVersion (firmwareFileName: *string*`,` **out result:** *int32*`)`**:** **void {raises-exception, use after open, enable}**	1.16
resetStatistics (statisticsBuffer: *string*`)`**:** **void {raises-exception, use after open, enable}**	1.16
retrieveStatistics (inout statisticsBuffer: *string*`)`**:** **void {raises-exception, use after open, enable}**	1.16
updateFirmware (firmwareFileName: *string*`)`**:** **void {raises-exception, use after open, enable}**	1.16
updateStatistics (statisticsBuffer: *string*`)`**:** **void {raises-exception, use after open, enable}**	1.16

Specific

Name

readFrame (frameData: *string***):** **void {raises-exception, use after open, claim, enable}**	1.16
startVideoRecording (fileName: *string***, overwrite:** *boolean***, recordingTime:** *int32***):** **void {raises-exception, use after open, claim, enable}**	1.16
stopVideoRecording ( ): **void {raises-exception, use after open, claim, enable}**	1.16
takePhotograph (fileName: *string***, overwrite:** *int32***):** **void {raises-exception, use after open, claim, enable}**	1.16

Events (UML interfaces)

Name	*Type*	*Mutability*	*Version*
upos::events::DataEvent			1.16
Status:	*int32*	{read-only}	
upos::events::DirectIOEvent			1.16
EventNumber:	*int32*	{read-only}	
Data:	*int32*	{read-write}	
Obj:	*object*	{read-write}	
upos::events::ErrorEvent			1.16
ErrorCode:	*int32*	{read-only}	
ErrorCodeExtended:	*int32*	{read-only}	
ErrorLocus:	*int32*	{read-only}	
ErrorResponse	*int32*	{read-write}	
upos::events::OutputCompleteEvent		Not Supported	
upos::events::StatusUpdateEvent			1.16
Status:	*int32*	{read-only}	

General Information

The Video Capture Device name is “VideoCapture”.

Capabilities

Video capture device class has the following capabilities:

- Get the captured frame data.
- Take a photograph and record it in a file.
- Take a movie and record it in a file.
- Read the encoded data from the bar code label.
- Detect the objects such as faces.

Video Capture Class Diagram

The following diagram shows the relationships between the Video Capture classes.

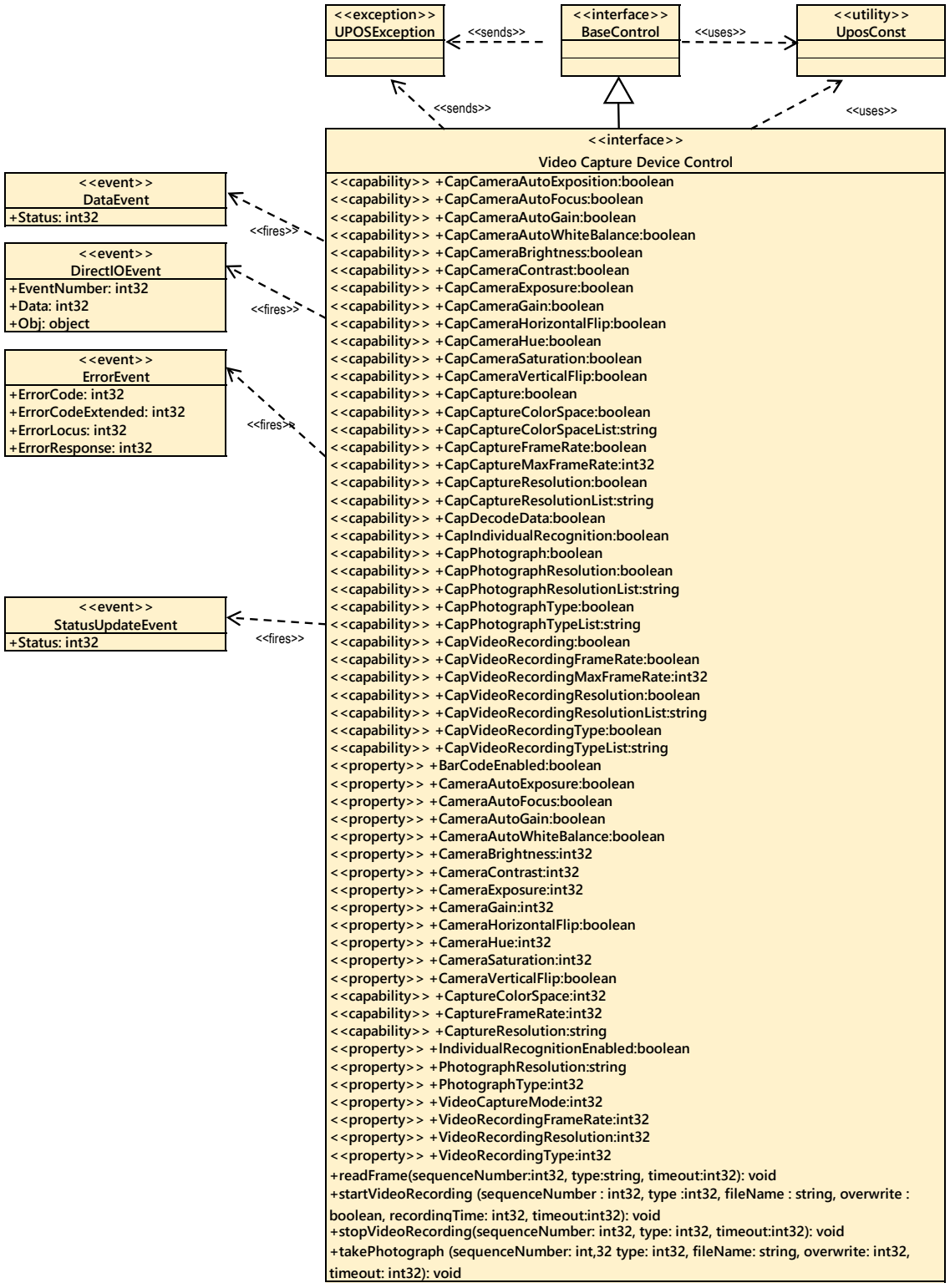


Fig. Chap. 39-1 Video Capture Class Diagram

Model

When video capture is enabled, the capture begins and the frame data can be retrieved by calling the **readFrame** method.

The resolution and frame rate of the frame data to be acquired depend on the operation mode set in the **VideoCaptureMode** property.

The following shows the setting to refer to each operation mode and the property for confirming valid values.

Capture only mode

Color space: **CaptureColorSpace** property
=> Valid value confirmation with **CapCaptureColorSpaceList** property

Resolution: **CaptureResolution** property => Valid value confirmation with **CapCaptureResolutionList** property

Frame rate: **CaptureFrameRat**e property

=> Valid value confirmation with **CapCaptureMaxFrameRate** property

Photo shooting mode

Color space: **CaptureColorSpace** property

=> Valid value confirmation with **CapCaptureColorSpaceList** property

Resolution: **PhotographResolution** property

=> Valid value confirmation with **CapPhotographResolutionList** property

Frame rate: **CaptureFrameRate** property

=> Valid value confirmation with **CapCaptureMaxFrameRate** property

Remarks: You can take pictures with **takePhotograph** method only in this mode.

Movie shooting mode

Color space: **CaptureColorSpace** property

=> Valid value confirmation with **CapCaptureColorSpaceLis**t property

Resolution: **VideoRecordingResolution** property

=> Valid value confirmation with the **CapVideoRecordingResolutionLis**t property

Frame rate: **VideoRecordingFrameRate** property

=> Valid value confirmation with **CapVideoRecordingMaxFrameRate** property

Remarks: It is possible to shoot movies with the **startVideoRecording** method only in this mode. Since the captured image / movie file is recorded in the area managed by the **"hard total"** service, the application must also support "**hard total**" service.

Input Model

Video capture control follows a common input model of event driven input, although there are some differences.

"**Control**" raises a **DataEvent** event when the recording started by the **startVideoRecording** method. And it ends when the specified time elapses and the recording to the specified file is completed.

When an application calls the **stopVideoRecording** method to end recording, **DataEvent** event will not occur. "

Also, by activating the **FaceCatchEnabled** property, face recognition is started, and even when a face is recognized, a **DataEvent** event is generated.

To distinguish between Recording Completed to File by Recording and **DataEvent** event of Face Recognition, refer to the **DataEventType** property.

The control sets VCP_ET_VIDEO when recording to the file by recording is completed, and sets VCP_ET_FACECATCH to the **DataEventType** property when recognizing the face. "

If the **AutoDisable** property is true, control will be disabled automatically when queuing **DataEvent** event.

If the **DataEventEnabled** property is true, the queued **DataEvent** is notified to the application. Just before triggering this event, the control copies the data to the property and sets the **DataEventEnabled** property to false to prevent further data events firing. This allows the control to queue subsequent input data while the application is processing the current input and processing the related properties. When the application finishes processing the current input data and is ready for the next data processing, setting the **DataEventEnabled** property to true will notify the **Data Event** again.

If an error occurs in the control while reading or processing the input data, an **ErrorEvent** is issued, and if the **DataEventEnabled** property is true, the application is notified.

By reading the **DataCount** property you get the number of **Data Events** queued by the control.

All input data queued in the control can be deleted by calling the **clearInput** method.

All data properties entered by **DataEvent** or **ErrorEvent** occurrence can be restored to the default value by calling the **clearInputProperties** method.

Bar Code Scan

By setting the **BarcodeEnabled** property to true for video capture, it is possible to scan the bar code by the camera.

When reading data from the bar code, the **DataEvent** event is queued in the scanner service object.

Scanned data is stored in the **ScanData** property. If the application sets the **DecodeData** property to true, the data is decoded to ScanDataLabel and ScanDataType.

Individual Recognition

By setting the **IndividualRecognitionEnabled** property to true for video capture, it is possible for objects to be recognized by the camera.

When an object is detected, a **DataEvent** is queued in the object recognition service object.

The detected data is stored in the **IndividualRecognitionInformation** and **IdividualIDs** of Individual Recognition Device properties.

Device Sharing

Video capture is an exclusive-use device, as follows:

- The application must claim the device before enabling it.
- The application must claim and enable the device before accessing many video capture-specific properties.
- The application must claim and enable the device before calling methods that manipulate the device.
- See the “Summary” table for precise usage prerequisites.

Properties (UML attributes)

BarCodeEnabled Property

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

Remarks If true, bar code scan is enabled. If false, bar code scan is disabled.
This property is initialized to false by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.
Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	Bar code scanning function is not supported (If it is set true)

See also **CapDecodeData** Property

CameraAutoExposure Property

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

Remarks If true, auto exposure of camera is enabled.
If false, auto exposure of camera is disabled.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.
Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	An invalid value was specified. Or it does not support this function.

See also **CapCameraAutoExposition** Property

CameraAutoFocus Property

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

Remarks If true, auto focus of camera is enabled.
If false, auto focus of camera is disabled.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.
Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	An invalid value was specified. Or it does not support this function.

See also **CapCameraAutoFocus** Property

CameraAutoGain Property

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

Remarks If true, auto gain of camera is enabled.
If false, auto gain of camera is disabled.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.
Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	An invalid value was specified. Or it does not support this function.

See also **CapCameraAutoGain** Property

CameraAutoWhiteBalance Property

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

Remarks If true, auto white balance of camera is enabled.
If false, auto white balance of camera is disabled.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.
Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	An invalid value was specified. Or it does not support this function.

See also **CapCameraAutoWhiteBalance** Property

CameraBrightness property

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

Remarks Indicate the brightness of camera.
Valid values range from 0 to 100.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.
Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	An invalid value was specified. Or it does not support this function.

See Also **CapCameraBrightness** Property

CameraContrast Property

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

Remarks Indicate the contrast of the camera.
Valid values range from 0 to 100.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.
Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	An invalid value was specified. Or it does not support this function.

See Also **CapCameraContrast** Property

CameraExposure Property

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

Remarks Indicate the exposure of camera. Valid values range from 0 to 100.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.
Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	An invalid value was specified. Or it does not support this function.

See also CapCameraExposure Property

CameraGain Property

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

Remarks Indicate the gain of camera. Valid values range from 0 to 100.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.
Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	An invalid value was specified. Or it does not support this function.

See also CapCameraGain Property

CameraHorizontalFlip Property

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

Remarks If true, horizontal flip of camera is enabled.
If false, horizontal flip of camera is disabled.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.
Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	An invalid value was specified. Or it does not support this function.

See Also **CapCameraHorizontalFlip** property

CameraHue Property

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

Remarks Indicate the hue of camera. Valid values range from 0 to 100.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.
Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	An invalid value was specified. Or it does not support this function.

See also **CapCameraHue** Property

CameraSaturation Property

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

Remarks Indicate the saturation of camera.
Valid values range from 0 to 100.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.
Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	An invalid value was specified. Or it does not support this function.

See also **CapCameraSaturation** Property

CameraVerticalFlip Property

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

Remarks If true, vertical flipping of the camera is enabled.
If false, vertical flipping of camera is disabled.
This property is initialized by the open method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.
Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	An invalid value was specified. Or it does not support this function.

See also **CapCameraVerticalFlip** Property

CapCameraAutoExposition Property

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

Remarks If true, can change the auto exposition of camera.
If false, cannot change the auto exposition of camera.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.

See also **CameraAutoExposition** Property

CapCameraAutoFocus Property

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

Remarks If true, can change the auto focus of camera.
If false, cannot change the auto focus of camera.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.

See also **CameraAutoFocus** Property

CapCameraAutoGain Property

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

Remarks If true, automatic gain change of the camera is possible.
If false, automatic gain change of camera is not possible.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.

See also **CameraAutoGain** Property

CapCameraAutoWhiteBalance Property

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

Remarks If true, auto white balance of camera is possible.
If false, auto white balance of camera is not possible.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see “**Errors**" on page Intro-20.

See also **CameraAutoWhiteBalance** Property

CapCameraBrightness Property

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

Remarks If true, the brightness of camera can be changed.
If false, the brightness of the camera cannot be changed.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see “**Errors**" on page Intro-20.

See also **CameraBrightness** Property

CapCameraContrast Property

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

Remarks If true, can change the contrast of camera.
If false, cannot change the contrast of camera.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see “**Errors**" on page Intro-20.

See also **CameraContrast** Property

CapCameraExposure Property

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

Remarks If true, can change the exposure of camera.
If false, cannot change the exposure of camera.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see “**Errors**" on page Intro-20.

See also **CameraExposure** Property

CapCameraGain Property

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

Remarks If true, can change the gain of camera.
If false, cannot change the gain of camera.
This property is initialized by the open method.

Errors A UposException may be thrown when this property is accessed.
For further information, see “**Errors**" on page Intro-20.

See also **CameraGain** Property

CapCameraHorizontalFlip Property

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

Remarks If true, can change the horizontal flip of camera.
If false, cannot change the horizontal flip of camera.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see “**Errors**" on page Intro-20.

See also **CameraHorizontalFlip** Property

CapCameraHue Property

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

Remarks If true, the hue of the camera can be changed.
If false, hue of the camera cannot be changed.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see “**Errors**" on page Intro-20.

See also **CameraHue** Property

CapCameraSaturation Property

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

Remarks If true, can change the saturation of camera.
If false, cannot change the saturation of camera.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see “**Errors**" on page Intro-20.

See also **CameraSaturation** Property

CapCameraVerticalFlip Property

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

Remarks If true, can change the vertical flip of camera.
If false, cannot change the vertical flip of camera.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.

See also **CameraVerticalFlip** Property

CapCapture Property

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

Remarks If true, it supports the capture function and can call the **readFrame** method and retrieve the frame data.
If false, it does not support the capture function and cannot retrieve the frame data. This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.

See also **readFrame** Method

CapCaptureColorSpace Property

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

Remarks If true, can change the capture color space.
If false, cannot change the capture color space.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.

CapCaptureColorSpaceList Property

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

Remarks Color space information supported by the device is indicated in a comma-separated list. Each color space information is composed of the following information and is shown in the following order separated by a colon (":").
This property is initialized by the **open** method.

Parameter	Description
Color space ID	ID for identifying the color space of RGB, YUV 422, etc.
Depth	Number of bits per 1 pixel

Errors A UposException may be thrown when this property is accessed.
For further information, see **"Errors"** on page Intro-20.

See also **CaptureColorSpace** Property

CapCaptureFrameRate Property

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

Remarks If true, can change the capture frame rate.
If false, cannot change the capture frame rate.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.

CapCaptureMaxFrameRate Property

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

Remarks Indicates the maximum frame rate that can be set for the **CaptureFrameRate** property.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further Information, see "**Errors**" on page Intro-20.

See also **CaptureFrameRate** Property

CapCaptureResolution Property

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

Remarks If true, capture resolution is enabled.
If false, capture resolution is disabled.
This property is initialized by the open method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.

See also **CaptureResolution** Property

CapCaptureResolutionList Property

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

Remarks Indicating the comma-separated list of possible resolutions for the **CaptureResolution** property. Resolution is indicated in "horizontal x height" format. For example, when you support 320x240, 640x480, 640x360, it is the following. "320 x 240, 640 x 480, 640 x 360".
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.

See also **CaptureResolution** Property

CapDecodeData Property

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

Remarks If true, the image scanner can read the bar code data.
The scanned bar code data is sent to the scanner service.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.

CapIndividualRecognition Property

Syntax **Cap Individual Recognition:** ***boolean*** **{read-only, access after open}**

Remarks If true, individual recognition function is supported.
If false, individual recognition function is not supported.
If this property is true, individual recognition can be done by setting **IndividualRecognitionEnabled** property to true.
If false, individual recognition cannot be performed.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see **"Errors"** on page Intro-20.

See also **IndividualRecognitionEnabled** Property

CapPhotograph Property

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

Remarks If true, photograph function is supported.
If false, photograph function is not supported.
If true, it is possible taking a photograph by calling the **takePhotograph** method. If false, it is not possible taking a photograph.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See also **takePhotograph** Method

CapPhotographResolution Property

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

Remarks If true, it is possible changing the photograph resolution.
If false, it is not possible changing the photograph resolution.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see **"Errors"** on page Intro-20.

CapPhotographResolutionList Property

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

Remarks A comma-separated list of possible resolutions for **PhotographResolution** property.
Resolution is indicated by Syntax "Horizontal x Vertical".
For example, when you support 320x240, 640x480, 640x360, it is the following. "320x240,640x480,640x360"
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see **"Errors"** on page Intro-20.

See also **PhotographResolution** Property

CapPhotographType Property

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

Remarks If true, photograph type can be changed.
If false, photograph type cannot be changed.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see **"Errors"** on page Intro-20.

CapPhotographTypeList Property

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

Remarks A comma-separated list of image format values that can be set for the **PhotographType** property.
For example, when supporting BMP and JPEG, it is the following.
"BMP, JPEG"

Note: The notation contents may be different depending on the device.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see **"Errors"** on page Intro-20.

See also **PhotographType** Property

CaptureColorSpace Property

Syntax **CaptureColorSpace:** ***string*** **{read-write, access after open}**

Remarks Indicates the color space ID of the frame data to be acquired by the **readFrame** method. Valid values are one of the values listed in the **CapCaptureColorSpaceList** property.
This property is referred to regardless of which operation mode is set by **VideoCaptureMode** property.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see **"Errors"** on page Intro-20.
Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	An invalid value was specified.

See also **CapCaptureColorSpaceList** Property、**VideoCaptureMode** Property、**readFrame** Method

CaptureFrameRate Property

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

Remarks Indicates the frame rate of frame data to be acquired by the **readFrame** method. Valid values range from 1 to **CapCaptureMaxFrameRate** property. This property is only referenced when VCP_VCM_CAPTURE is set in **VideoCaptureMode** property.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see **"Errors"** on page Intro-20.

Value	Meaning
E_ILLEGAL	An invalid value was specified.

See also **CapCaptureMaxFrameRate** Property, **VideoCaptureMode** Property、**readFrame** Method

CaptureResolution Property

Syntax **CaptureResolution:** ***string*** **{read-write, access after open}**

Remarks Indicates the resolution of the frame data acquired by the **readFrame** method.
Valid values are one of those listed in **CapCaptureResolutionList** property.
This property is only referenced when VCP_VCM_CAPTURE is set in **VideoCaptureMode** property.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see **"Errors"** on page Intro-20.
Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	An invalid value was specified.

See also **CapCaptureResolutionList** Property、**VideoCaptureMode** Property、**readFrame** Method

CapVideoRecording Property

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

Remarks If true, video recording function is supported.
If false video recording function is not supported.
If this property is true, movie recording can be done by calling the **startVideoRecording** method.
If false, movie recording cannot be performed.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see **"Errors"** on page Intro-20.

See also **StartVideoRecording** Property

CapVideoRecordingFrameRate Property

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

Remarks If true, video recording frame rate can be changed.
If false, video recording frame rate cannot be changed.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see **"Errors"** on page Intro-20.

CapVideoRecordingMaxFrameRate Property

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

Remarks Indicates the maximum frame rate that can be set in **VideoRecordingFrameRate** property.
This property is initialized by the open method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.

See also **VideoRecordingFrameRate** Property

CapVideoRecordingResolution Property

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

Remarks If true, video recording resolution can be changed.
If false, video recording resolution cannot be changed.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see **"Errors"** on page Intro-20.

CapVideoRecordingResolutionList Property

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

Remarks A comma-separated list of possible resolutions for the **VideoRecordingResolution** property.
Resolution is indicated by "Horizontal x Vertical" format.
For example, when it supports 320x240, 640x480, 640x360, it is the following. "320x240,640x480,640x360"
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see **"Errors"** on page Intro-20.

See also **VideoRecordingResolution** Property

CapVideoRecordingType Property

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

Remarks If true, video recording type can be changed.
If false, video recording type cannot be changed.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see **"Errors"** on page Intro-20.

CapVideoRecordingTypeList Property

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

Remarks A comma-separated list of image format values that can be set for the **VideoRecordingType** property.
For example, when AVI_IYUV, AVI_MJPG is supported, it is the following. "AVI_IYUV, AVI_MJPG"
Note: The notation contents may be different depending on the device.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see **"Errors"** on page Intro-20.

See also **VideoRecordingType** Property

IndividualRecognitionEnabled Property

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

Remarks If true individual recognition is enabled.
If false, individual recognition is disabled.
This property is initialized to false by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.
Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	Individual recognition function is not supported (If it is set true)

See also **CapIndividual Recognition** Property

PhotographResolution Property

Syntax **PhotographResolution:** ***string*** **{read-write, access after open}**

Remarks It shows the resolution of the frame data acquired by the **readFrame** method and the photograph taken with the **takePhotograph** method.
Valid values are one of those listed in **CapPhotographResolutionList** property.
This property is referenced only when VCP_VCM_PHOTO is set in **VideoCaptureMode** property.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.
Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	An invalid value was specified.

See also **CapPhotographResolutionList** Property, **VideoCaptureMode** Property, **readFrame** Method, **takePhotograph** Method

PhotographType Property

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

Remarks Indicates the image format of photos taken with the **takePhotograph** method.
Valid values are one of the values listed in the **CapPhotographTypeList** property.
This property is referenced only when VCP_VCM_PHOTO is set in **VideoCaptureMode** property.
This property is initialized by the **open** method.

Remarks Indicates the image format of photos taken with the **takePhotograph** method.
Valid values are one of the values listed in the **CapPhotographTypeList** property.
This property is referenced only when VCP_VCM_PHOTO is set in **VideoCaptureMode** property.
This property is initialized by the **open** method

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.
Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	An invalid value was specified.

See also **CapPhotographTypeList** Property、**takePhotograph** Method

VideoCaptureMode Property

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

Remarks Indicate the operation mode of video capture.
Valid values are as follows

Parameter	Description
VCP_VCMODE_CAPTURE	This mode is for capture only. The values of the **CaptureColorSpace, CaptureResolution,** and **aptureFrameRate** properties are applied to the color space, resolution, and frame rate of frame data that can be acquired with the **readFrame** method**.**
VCP_VCMODE_PHOTO	This mode is for capture and taking photograph. The values of the **CaptureColorSpace** and **CaptureFrameRate** properties are applied to the color space and frame rate of the frame data that can be acquired by the **readFrame** method, and the resolution is applied to the resolution of the **CapPhotographResolution** property.
VCP_VCMODE_VIDEO	This mode is for capture and movie shooting. The value of the **CaptureColorSpace** property is applied to the color space of the frame data that can be acquired by the **readFrame** method, the values of the **CapVideoRecordingResolution** property and the C**apVideoRecordingFrameRate** property are applied to the resolution and the frame rate. This property is initialized to VCP_VCMODE_CAPTURE by the **open** method. Indicate the operation mode of video capture.

Errors A UposException may be thrown when this property is accessed.
For further information, see “**Errors**" on page Intro-20.

See also **CaptureColorSpace** Property、*CaptureResolution* Property, **CaptureFrameRate** Property、**CapPhotographResolution** Property、**CapVideoRecordingResolution** Property、**CapVideoRecordingFrameRate** Property、**readFrame** Method

VideoRecordingFrameRate Property

Syntax **VideoRecordingFrameRate;** ***int32*** **{read-write, access after open}**

Remarks Indicates the frame rate of the frame data acquired by the **readFrame** method and the movie taken with the **startVideoRecording** method. Valid values range from 1 to **CapVideoRecordingMaxFrameRate** property.
This property is only referred when VCP_VCM_VIDEO is set in **VideoCaptureMode** property.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see **"Errors"** on page Intro-20.
Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	An invalid value was specified.

See also **CapVideoRecordingMaxFrameRate** Property、**VideoCaptureMode** Property、**readFrame** Method, **startVideoRecording** Method

VideoRecordingResolution Property

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

Remarks Indicates the resolution of the frame data acquired by the **readFrame** method and the photograph taken with the **startVideoRecording** method. Valid values are one of the values listed in the **CapVideoRecordingResolutionList** property.
This property is only referred when VCP_VCM_VIDEO is set in **VideoCaptureMode** property.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see **"Errors"** on page Intro-20.
Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	An invalid value was specified.

See also **CapVideoRecordingResolutionList** Property, **VideoCaptureMode** Property、**readFrame** Method **startVideoRecording** Method

VideoRecordingType Property

Syntax **VideoRecordingType; *string* {read-write, access after open}**

Remarks Indicate the shape of the movie taken with the **startVideoRecording** method.
Valid values are one of those listed in **CapVideoRecordingTypeList** property.
This property is only referred when VCP_VCM_VIDEO is set in **VideoCaptureMode** property.
This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed.
For further information, see **"Errors"** on page Intro-20.
Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	An invalid value was specified.

See also **CapVideoRecordingTypeList** Property,
startVideoRecording Method

Methods (UML operations)

readFrame Method

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

Parameter	Description
frameData	Indicates the area where frame data is stored.

Remarks Acquires the captured frame data and stores it in frameData.
The color space and resolution of frame data differs depending on the operation mode set in the **VideoCaptureMode** property.
For details, refer to the **VideoCaptureMode** property.
This method is executed synchronously.

Errors A UposException may be thrown when this method is invoked.
For further information, see "**Errors**" on page Intro-20.
Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	This function is not supported

See also **VideoCaptureMode** Property

startVideoRecording method

Syntax **startVideoRecording (fileName : string, overwrite: *boolean*, recordingTime: *int32*):**
void{raises-exception, use after open, claim, enable}

Parameter	Description
filename	Specify the name of the movie file to be recorded.
Overwrite	Specify the behavior when the same name file exists. If true, it is overwritten. If false, it will raise the UposException.
recordingTime	Specify the time for recording in seconds. If FOREVER (-1) is specified, recording will continue until the **stopVideoRecording** method is called.

Remarks Recording starts with the setting contents of the **CaptureColorSpace** and **VideoRecordingResolution** properties, and recording starts in the format set by the **VideoRecordingType** property.
This method is executed asynchronously.
When the time specified in RecordingTime has elapsed, or by calling the **stopVideoRecording** method, recording is completed and the movie file specified by fileName is recorded.
Also, S_BUSY is set in the **Status** property during movie execution.
The place where video files are recorded is the area managed by "hard total" service.

Errors A UposException may be thrown when this method is invoked.
For further information, see "**Errors**" on page Intro-20.
Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	fileName is too long or contains characters that cannot be used, or 0 is specified for recordingTime.
E_EXISTS	fileName already exists. (If overwrite is false)
E_BUSY	Cannot execute because it is recording.

See also **CaptureColorSpace** Property、**VideoRecordingResolution** Property、**VideoRecordingType** Property、**stopVideoRecording** Method

stopVideoRecording method

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

Remarks The recording process started by the **startVideoRecording** method has ended and the recording of the movie image file is completed.

Errors A UposException may be thrown when this method is invoked.
For further information, see "**Errors**" on page Intro-20.
Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	It is not recorded.

See also **startVideoRecording** Method

takePhotograph Method

Syntax **takePhotograph (fileName: *string*, overwrite: *int32*, timeout: *int32*):**
void{raises-exception, use after open, claim, enable}

Parameter	Description
fileName	Specify the image file name to be recorded.
overwrite	Specify the behavior when the same name file exists. If true it overwrites. If false, UposException is thrown.

Remarks Take photos with setting contents of **CaptureColorSpace** property, **PhotographResolution** property, **PhotographType** property and record images. Before calling this method, it needs to set the **VideoCaptureMode** property to VCP_VCM_PHOTO and change to the photo shooting mode. This method is executed synchronously.
The location where image files are recorded is the area managed by "hard total" service.

Errors A UposException may be thrown when this method is invoked.
For further information, see "**Errors**" on page Intro-20.
Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	One of the following occurred. FileName is too long or contains unusable characters. **VideoCaptureMode** property is not VCM_PHOTO
E_EXISTS	fileName already exist. (When overwrite=false)

See also **VideoCaptureMode** Property、**CaptureColorSpace** Property、
PhotographResolution Property、**PhotographType** Property

Events (UML interfaces)

DataEvent

<<event>> **upos::events::DataEvent**

Status:*int32*{read-only}

Description Notifies the application when data from the Video Capture device is available to be read.

Attributes This event contains the following attributes:

Attribute	Type	Description
Status	*int32*	*Set to 0.*

Remarks Before this event is delivered, the Video Capture movie image is placed into **readFrame.**

This event is to be used only for those types of vendor specific functions that are not otherwise described.

Use of this event may restrict the application program programform being used with other vendor's devices which may not have any knowledge of the Service's need for this event.

See Also **"Events"** on page Intro-19, **directIO** method

DirectIOEvent

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

EventNumber : ***int32*** **{read-only}**
Data : ***int32*** **{read-write}**
Obj : ***object*** **{read-write}**

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

Attributes This event contains the following attributes:

Attribute	Type	Description
EventNumber	*int32*	Event number whose specific values are assigned by the Service.
Data	*int32*	Additional numeric data. Specific values vary by the *EventNumber* and the Service. This attribute is settable.
Obj	*object*	Additional data whose usage varies by the *EventNumber* and the Service. This attribute is settable.

Remarks This event is to be used only for those types of vendor specific functions that are not otherwise described.

Use of this event may restrict the application program programform being used with other vendor's devices which may not have any knowledge of the Service's need for this event.

See Also **"Events"** on page Intro-19, **directIO** method

ErrorEvent

<<event>> **upos::events::ErrorEvent**

ErrorCode : ***int32*** **{read-only}**
ErrorCodeExtended : ***int32*** **{read-only}**
ErrorLocus : ***int32*** **{read-only}**
ErrorResponse : ***int32*** **{read-write}**

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

Attributes This event contains the following attributes:

Attributes	Type	Description
ErrorCode	*int32*	Error code causing the error event. See a list of Error Codes on page 20.
ErrorCodeExtended	*int32*	Extended Error code causing the error event. If *ErrorCode is* E_EXTENDED, then see values below. Otherwise, it may contain a Service-specific value.
ErrorLocus	*int32*	Location of the error. If EL_OUTPUT is specified. An error occurred during asynchronous action.
ErrorResponse	*int32*	Pointer to the error event response. See *ErrorResponse* below for values.

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

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

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

Value	Meaning
ER_RETRY	Retry sending the data. The error state is exited. May be valid for some input devices when the locus is EL_INPUT, in which case the input is retried and the error state is exited. Typically, valid for asynchronous output devices when the locus is EL_OUTPUT, in which case the asynchronous output is retried and the error state is exited. This is the default response when the locus is EL_OUTPUT.
ER_CLEAR	Valid for all loci: EL_INPUT, EL_INPUT_DATA, and EL_OUTPUT. Clear all buffered input or output data

(including all asynchronous output). The error state is exited. This is the default response when the locus is EL_INPUT.

ER_CONTINUEINPUT

Only valid when the locus is EL_INPUT_DATA. Acknowledges that a data error has occurred and directs the Device to continue input processing. The Device remains in the error state and will deliver additional **DataEvent**s as directed by the **DataEventEnabled** property. When all input has been delivered and **DataEventEnabled** is again set to true, then another **ErrorEvent** is delivered with locus EL_INPUT. This is the default response when the locus is EL_INPUT_DATA.

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

Unlike a **DataEvent**, the Device does not disable further **DataEvent**s or input **ErrorEvent**s; it leaves the **DataEventEnabled** property value at true. Note that the application may set **DataEventEnabled** to false within its event handler if subsequent input events need to be disabled for a period of time.

See Also **"Device Input Model"** on page Intro-22, **"Error Handling"** on page Intro-23, **"Device Output Models"** on page Intro-25.

StatusUpdateEvent

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

Description *Notifies the application that there is a change in the power status of the Video Capture device.*

Attributes This event contains the following attribute:

Attributes	Type	Description
Status	*int32*	Indicates a change in the power status of the unit.

Note that Release 1.3 added Power State Reporting with additional *Power reporting* **StatusUpdateEvent** *values.*

The Update Firmware capability added additional *Status* values for communicating the status/progress of an asynchronous update firmware process. See "**StatusUpdateEvent**" description on page 1-34.

CHAPTER 40

Individual Recognition

This Chapter defines the Individual Recognition device category.

Summary

Properties (UML attributes)

Common	*Type*	*Mutability*	*Version*	*May Use After*
AutoDisable:	*boolean*	{read-write}	1.16	open
CapCompareFirmwareVersion:	*boolean*	{read-only}	1.16	open
CapPowerReporting:	*int32*	{read-only}	1.16	open
CapStatisticsReporting:	*boolean*	{read-only}	1.16	open
CapUpdateFirmware:	*boolean*	{read-only}	1.16	open
CapUpdateStatistics:	*boolean*	{read-only}	1.16	open
CheckHealthText:	*string*	{read-only}	1.16	open
Claimed:	*boolean*	{read-only}	1.16	open
DataCount:	*int32*	{read-only}	1.16	open
DataEventEnabled:	*boolean*	{read-write}	1.16	open
DeviceEnabled:	*boolean*	{read-write}	1.16	open, claim
FreezeEvents:	*boolean*	{read-write}	1.16	open
OutputID:	*int32*	{read-only}	1.16	Not Supported
PowerNotify:	*int32*	{read-write}	1.16	open
PowerState:	*int32*	{read-only}	1.16	open
State:	*int32*	{read-only}	1.16	--
DeviceControlDescription:	*string*	{read-only}	1.16	--
DeviceControlVersion:	*int32*	{read-only}	1.16	--
DeviceServiceDescription:	*string*	{read-only}	1.16	open
DeviceServiceVersion:	*int32*	{read-only}	1.16	open
PhysicalDeviceDescription:	*string*	{read-only}	1.16	open
PhysicalDeviceName:	*string*	{read-only}	1.16	Open

Specific	*Type*	*Mutability*	*Version*	*May Use After*
CapIndividualList:	*string*	{read-only}	1.16	open
IndividualRecognitionFilter	*string*	{read-writer}	1.16	open
IndividualRecognitionInformation	*string*	{read-only}	1.16	open
IndividualIDs:	*string*	{read-write}	1.16	open, claim & enable

Methods (UML operations)

Common

Name	*Version*
open (logicalDeviceName: *string*`)`**:** **void {raises-exception}**	1.16
close ( ): **void {raises-exception, use after open}**	1.16
claim (timeout: *int32*`)`**:** **void {raises-exception, use after open}**	1.16
release ( ): **void {raises-exception, use after open, claim}**	1.16
checkHealth (level: *int32*`)`**:** **void {raises-exception, use after open, enable}**	1.16
clearInput ( ): **void { }**	Not supported
clearInputProperties ( ): **void { }**	Not supported
clearOutput ( ): **void { }**	Not supported
compareFirmwareVersion (firmwareFileName: *string*, **out result:** *int32*`)`**:** **void {raises-exception, use after open, enable}**	1.16
directIO (command: *int32*, **inout data:** *int32*, **inout obj:** *object*`)`**:** **void {raises-exception, use after open}**	1.16
resetStatistics (statisticsBuffer: *string*`)`**:** **void {raises-exception, use after open, enable}**	1.16
retrieveStatistics (inout statisticsBuffer: *string*`)`**:** **void {raises-exception, use after open, enable}**	1.16
updateFirmware (firmwareFileName: *string*`)`**:** **void {raises-exception, use after open, enable}**	1.16
updateStatistics (statisticsBuffer: *string*`)`**:** **void {raises-exception, use after open, enable}**	1.16

Events (UML interfaces)

Name	*Type*	*Mutability*	*Version*
upos::events::DataEvent			1.16
Status:	*int32*	{read-only}	
upos::events::DirectIOEvent			1.16
EventNumber:	*int32*	{read-only}	
Data:	*int32*	{read-write}	
Obj:	*object*	{read-write}	
upos::events::ErrorEvent			1.16
ErrorCode:	*int32*	{read-only}	
ErrorCodeExtended:	*int32*	{read-only}	
ErrorLocus:	*int32*	{read-only}	
ErrorResponse:	*int32*	{read-write}	
upos::events::OutputCompleteEvent		***Not Supported***	1.16
upos::events::StatusUpdateEvent			1.16
Status:	*int32*	{read-only}	

General Information

The Individual Recognition programmatic name is “Individual Recognition”.

Capabilities

The Individual Recognition has the following set of capabilities:

Analyzes the image of the camera and recognizes Individuals such as people and balls.

Individual Recognition Class Diagram

The following diagram shows the relationships between the Individual Recognition classes.

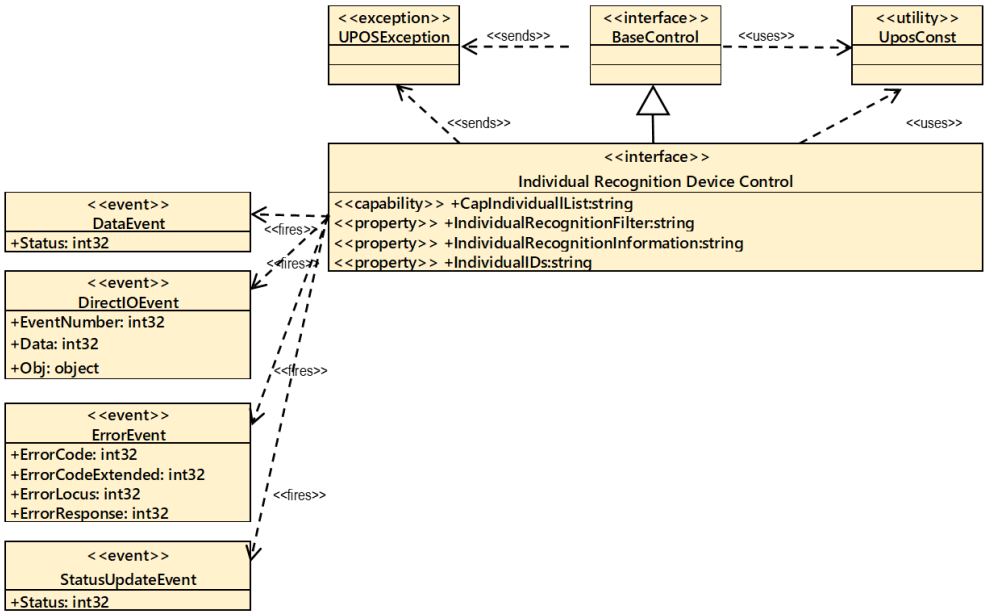


Fig. Chap.40-1 Individual Recognition Class Diagram

Model

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

Input Model

The **readValue** method follows the UnifiedPOS Input model.

- If the **AutoDisable** property is true, then the device automatically disables itself when a **DataEvent** is enqueued.
- An enqueued **DataEvent** can be delivered to the application when the **DataEventEnabled** property is true and other event delivery requirements are met. Just before delivering this event, data is copied into corresponding properties, and further data events are disabled by setting **DataEventEnabled** to false. This causes subsequent input data to be enqueued while the application processes the current input and associated properties. When the application has finished processing the current input and is ready for more data, it reenables events by setting **DataEventEnabled** to true.
- An **ErrorEvent** (or events) is enqueued if an error occurs while gathering or processing input, and is delivered to the application when **DataEventEnabled** is true and other event delivery requirements are met.
- The **DataCount** property may be read to obtain the total number of enqueued **DataEvent**s.
- All enqueued input may be deleted by calling **clearInput**. See the **clearInput** method description for more details.
- All data properties that are populated as a result of firing a **DataEvent** or **ErrorEvent** can be set back to their default values by calling the **clearInputProperties** method.
- Identifiable individuals are indicated by the **CapIndividualList** property.
- Check the functions supported by the device, set validity / invalidity, etc. with the **IndividualRecognitionInformation** property.
- Recognized data is stored in the **IndividualRecognitionInformation** property, **IndividualIDs**.

Device Sharing

The Individual Recognition is an exclusive-use device, as follows:

- The application must claim the device before enabling it.
- The application must claim and enable the device before the device begins reading input.
- See the “Summary” table for precise usage prerequisites.

Properties (UML attributes)

CapIndividualList Property

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

Remarks Recognizable Individual information is indicated by the list separated by a separator ",".

Each Individual information consists of the following information and is shown in the following order, separated with a colon (":").

Parameter	Meaning
IndividualID	An ID indicated an identifiable Individual
IndividualName	A Name of an Individual.

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

Errors A UposException may be thrown when this property is accessed. For further information, see “Errors" on page Intro-20.

See Also “**Individual**IDs" Property on page XX-11

IndividualRecognitionFilter Property

Syntax **IndividualRecognitionFilter:** ***string*** **{read-write, access after open}**

Remarks Holds data indicating the following.

Individual Recognition Function Information:

- Support for various functions (supported functions are defined by the device).
- Valid / invalid state of various functions.
- Types handled by various functions (e.g., "male" "female" in gender recognition, etc.).
- Filter setting of various functions.

All Individual recognition function information data is defined by the device. By referring to these contents, the application can determine the support scope etc. Thereby, the application can control each function by changing the valid / invalid state and / or the filter setting of various functions.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **“Errors"** on page Intro-20.

IndividualRecognitionInformation Property

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

Remarks Holds data indicating the following.
Individual recognition input data.
All Individual recognition input data is defined by the device.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.

IndividualIDs Property

Syntax **IndividualIDs:** ***string*** **{read-write, access after open}**

Remarks Holds an IndividualID recognized by Individual recognition and indicated by separated with a colon (":").

Its value is set prior to a **DataEvent** being delivered to the application.

Errors A UposException may be thrown when this property is accessed.
For further information, see "**Errors**" on page Intro-20.

See Also **CapIndividualList** Property

CHAPTER 41

Sound Recorder

This Chapter defines the Sound Recorder device category.

Summary

Properties(UML attributes)

Common	*Type*	*Mutability*	*Version*	*May Use After*
AutoDisable:	*boolean*	{read-write}	1.16	open
CapCompareFirmwareVersion:	*int32*	{read-only}	1.16	open
CapPowerReporting:	*boolean*	{read-only}	1.16	open
CapStatisticsReporting:	*boolean*	{read-only}	1.16	open
CapUpdateFirmware:	*boolean*	{read-only}	1.16	open
CapUpdateStatistics:	*string*	{read-only}	1.16	open
CheckHealthText:	*boolean*	{read-only}	1.16	open
Claimed:	*int32*	{read-only}	1.16	open
DataCount:	*boolean*	{read-only}	1.16	open
DataEventEnabled:	*boolean*	{read-write}	1.16	open
DeviceEnabled:	*boolean*	{read-write}	1.16	open, claim
FreezeEvents:	*int32*	{read-write}	1.16	open
OutputID:	*int32*	{read-only}	1.16	Not Supported
PowerNotify:	*int32*	{read-write}	1.16	open
PowerState:	*boolean*	{read-only}	1.16	open
State:	*boolean*	{read-only}	1.16	open
DeviceControlDescription:	*string*	{read-only}	1.16	--
DeviceControlVersion:	*int32*	{read-only}	1.16	--
DeviceServiceDescription:	*string*	{read-only}	1.16	open
DeviceServiceVersion:	*int32*	{read-only}	1.16	open
PhysicalDeviceDescription:	*string*	{read-only}	1.16	Open

Properties (Continued)

Common(continued)	*Type*	*Mutability*	*Version*	*May Use After*
PhysicalDeviceName:	*string*	{read-only}	1.16	open

Specific	*Type*	*Mutability*	*Version*	*May Use After*
CapChannel:	*boolean*	{read-only}	1.16	open
CapSamplingRate:	*boolean*	{read-only}	1.16	open
CapSoundType:	*boolean*	{read-only}	1.16	open
CapRecordingLevel:	*boolean*	{read-only}	1.16	open
CapChannelList:	*string*	{read-only}	1.16	open
CapSamplingRateList:	*string*	{read-only}	1.16	open
CapSoundTypeList:	*string*	{read-only}	1.16	open
Channel:	*string*	{read-write}	1.16	open, claim & enable
SamplingRate:	*string*	{read-write}	1.16	open, claim & enable
SoundType:	*string*	{read-write}	1.16	open, claim & enable
RecordingLevel:	*int32*	{read-write}	1.16	open, claim & enable

Methods(UML operations)

Common

Name	*Version*
open (logicalDeviceName: *string***):** **void {raises-exception}**	1.16
close ( ): **void {raises-exception, use after open}**	1.16
claim (timeout: *int32***):** **void {raises-exception, use after open}**	1.16
release ( ): **void {raises-exception, use after open, claim}**	1.16
checkHealth (level: *int32***):** **void {raises-exception, use after open, enable}**	1.16
clearInput ( ): **void { }**	Not supported
clearInputProperties ( ): **void { }**	Not supported

Methods (UML operations)(continued)

Common

Name	*Version*
clearOutput ( ): **void { }**	Not supported
compareFirmwareVersion (firmwareFileName: *string*, **out result:** *int32*): **void {raises-exception, use after open, enable}**	1.16
directIO (command: *int32*, **inout data:** *int32*, **inout obj:** *object*): **void {raises-exception, use after open}**	1.16
resetStatistics (statisticsBuffer: *string*): **void {raises-exception, use after open, enable}**	1.16
retrieveStatistics (inout statisticsBuffer: *string*): **void {raises-exception, use after open, enable}**	1.16
updateFirmware (firmwareFileName: *string*): **void {raises-exception, use after open, enable}**	1.16
updateStatistics (statisticsBuffer: *string*): **void {raises-exception, use after open, enable}**	1.16

Specific

Name	*Version*
startRecording (FileName: *string*, **OverWrite:** *boolean*, **RecordingTime:***int32*): **void {raises-exception, use after open, claim, enable}**	1.16
stopRecording ( ): **Void {raises-exception, use after open, claim, enable}**	1.16

Events (UML interfaces)

Name	*Type*	*Mutability*	*Version*
upos::events::DataEvent			1.16
Status:	*int32*	{read-only}	
upos::events::DirectIOEvent			1.16
EventNumber:	*int32*	{read-only}	
Data:	*int32*	{read-write}	
Obj:	*object*	{read-write}	
upos::events::ErrorEvent			1.16
ErrorCode:	*int32*	{read-only}	
ErrorCodeExtended:	*int32*	{read-only}	
ErrorLocus:	*int32*	{read-only}	
ErrorResponse:	*int32*	{read-write}	
upos::events::OutputCompleteEvent		***Not Supported***	1.16
upos::events::StatusUpdateEvent			1.16
Status:	*int32*	{read-only}	

General Information

The Sound Recorder programmatic name is "SoundRecorder".

Capabilities

The Sound Recorder has the following capability:

- Save the recorded sound to a file.

Sound Recorder Class Diagram

The following diagram shows the relationships between the Sound Recorder classes.

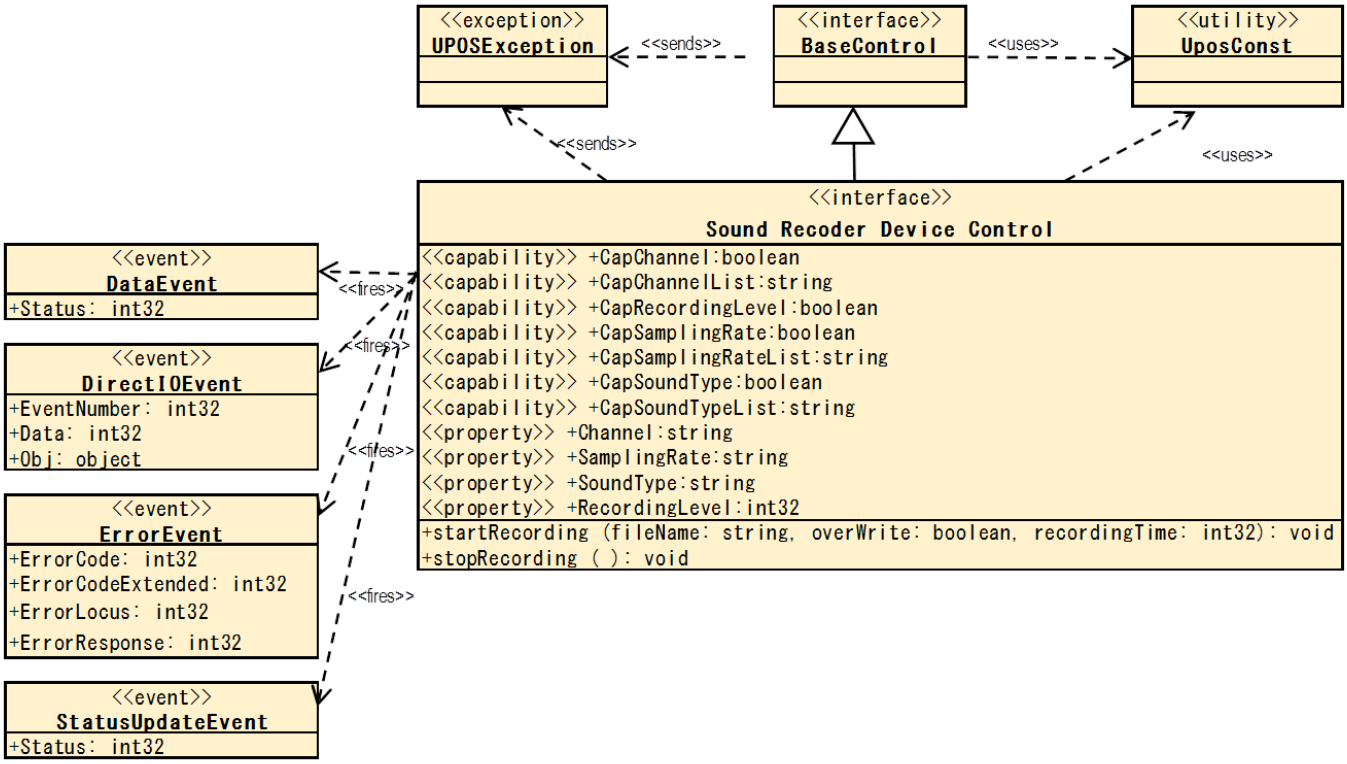


Fig. Chap. 41-1 Sound Recorder Class Diagram

Model

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

- "The control will generate a **DataEvent** when the recording started by the **startRecording** method ends when the specified time elapses and the recording to the specified file is completed.
- When an application calls the **stopRecording** method to end recording, **DataEvent** will not occur."
- If the **AutoDisable** property is true, then the device automatically disables itself when a **DataEvent** is enqueued.
- An enqueued **DataEvent** can be delivered to the application when the **DataEventEnabled** property is true and other event delivery requirements are met. Just before delivering this event, data is copied into corresponding properties, and further data events are disabled by setting **DataEventEnabled** to false. This causes subsequent input data to be enqueued while the application processes the current input and associated properties. When the application has finished processing the current input and is ready for more data, it reenables events by setting **DataEventEnabled** to true.
- An **ErrorEvent** (or events) is enqueued if an error occurs while gathering or processing input, and is delivered to the application when **DataEventEnabled** is true and other event delivery requirements are met.
- The **DataCount** property may be read to obtain the total number of enqueued **DataEvent**s.
- All enqueued input may be deleted by calling **clearInput**. See the **clearInput** method description for more details.
- All data properties that are populated as a result of firing a **DataEvent** or **ErrorEvent** can be set back to their default values by calling the **clearInputProperties** method.
- Since audio files are recorded in the area managed by the "hard total" service, the application must also support "hard total" services.

Device Sharing

The Sound Recorder is an exclusive-use device, as follows:

- The application must claim the device before enabling it.
- The application must claim and enable the device before accessing some properties or calling methods that update the device.
- See the “Summary” table for precise usage prerequisites.
- The image display mode of the graphics control is as follows.

Properties(UML attributes)

CapChannel Property

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

Remarks If true, the application can change the channel.

If false, the application cannot change the channel.

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

Errors UposException may be thrown when this property is accessed.
For further information, see “Errors" on page Intro-20.

See Also **Channel** Property

CapSamplingRate Property

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

Remarks If true, the application can change the sampling rate.

If false, the application cannot change the sampling rate.

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

Errors UposException may be thrown when this property is accessed.
For further information, see “Errors" on page Intro-20.

See Also **SamplingRate** Property.

CapSoundType Property

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

Remarks If true, the application can change the sound file type.

If false, the application cannot change the sound file type.

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

Errors UposException may be thrown when this property is accessed.
For further information, see “Errors" on page Intro-20.

See Also **SoundType** Property.

CapRecordingLevel Property

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

Remarks If true, the application can change the recording level.

If false, the application cannot change the recording level.

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

Errors UposException may be thrown when this property is accessed.
For further information, see “Errors" on page Intro-20.

See Also **CapRecordingLevel** Property.

CapChannelList Property

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

Remarks Contains the comma-delimited list of channel that is supported by the device.

For example, if the device only supports 1ch and 2ch and 4ch, then this property should be set to "1,2,4".

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

Errors UposException may be thrown when this property is accessed.
For further information, see “Errors" on page Intro-20.

See Also **Channel** Property.

CapSamplingRateList Property

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

Remarks Contains the comma-delimited list of sampling rate that are supported by the device.

For example, if the device only supports 44.1KHz and 48KHz and 96KHz, then this property should be set to "44100,48000,96000".

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

Errors UposException may be thrown when this property is accessed.
For further information, see “Errors" on page Intro-20.

See Also **SamplingRate** Property.

CapSoundTypeList Property

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

Remarks Contains the comma-delimited list of sound file type that is supported by the device.

For example, if the device only supports WAV and OGG, then this property should be set to "WAV,OGG".

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

Errors UposException may be thrown when this property is accessed.
For further information, see “Errors" on page Intro-20.

See Also **SoundType** Property.

Channel Property

Syntax **Channel : *string* {read-write, access after open, claim}**

Remarks Holds the channel during recording.

Valid values are one of the values listed in the **CapChannelList** property.

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

Errors UposException may be thrown when this property is accessed.
For further information, see “Errors" on page Intro-20.

Value	Meaning
E_ILLEGAL	An invalid value was specified.
E_BUSY	Property could not be set because it is recording.

See Also **CapChannel** Property、**CapChannelList** Property

SamplingRate Property

Syntax **SamplingRate : *string*{read-write, access after open, claim}**

Remarks Holds the sampling rate during recording.

Valid values are one of the values listed in the CapSamplingRateList property.

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

Errors UposException may be thrown when this property is accessed.
For further information, see “Errors" on page Intro-20.

Value	Meaning
E_ILLEGAL	An invalid value was specified.
E_BUSY	Property could not be set because it is recording.

See Also **CapSamplingRate** Property、**CapSamplingRateList** Property

SoundType Property

Syntax **SoundType : *string* {read-write, access after open, claim}**

Remarks Holds the audio file format to be recorded.

Valid values are one of the values listed in the **CapSoundTypeList** property.

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

Errors UposException may be thrown when this property is accessed.
For further information, see “Errors" on page Intro-20.

Value	Meaning
E_ILLEGAL	An invalid value was specified.
E_BUSY	Property could not be set because it is recording.

See Also **CapSoundType** Property、**CapSoundTypeList** Property

RecordingLevel Property

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

Remarks Holds the recording level during recording.

Legal values range from zero through 100.

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

Errors UposException may be thrown when this property is accessed.
For further information, see “Errors" on page Intro-20.

Value	Meaning
E_ILLEGAL	An invalid value was specified.

See Also **CapRecordingLevel** Property

Methods(UML operations)

startRecording Method

Syntax **startRecording (fileName :** *string*, **overWrite :** *boolean*, **recordingTime :** *int32*)**:** **void{raises-exception, use after open, claim, enable}**

Parameter	Description
fileName	Specify the file name of the image to be loaded.
overWrite	Specify the behavior when the same name file exists. If it is true it will be overwritten and if false it will return an error.
recordingTime	Specify the time for recording in seconds. If OPOS_FOREVER (-1) is specified, recording will continue until you call the **stopRecording** method.

Remarks Recording starts with the settings of the **Channel** property, **SamplingRate** property, and **RecordingLevel** property, and recording starts in the format set by SoundType.

Errors A UposException may be thrown when this method is invoked. For further information, see **"Errors"** on page Intro-20

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

Value	Meaning
E_ILLEGAL	FileName is too long or contains characters that cannot be used, or 0 is specified for RecordingTime.
E_EXISTS	FileName already exists. (When OverWrite is FALSE)
E_BUSY	It cannot be executed as it is recording.

See Also **Channel** Property、**SamplingRate** Property、**SoundType** Property、**RecordingLevel** Property、**stopRecording** Method

stopRecording Method

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

Remarks Finish the recording and complete the recording of the audio file.

Errors A UposException may be thrown when this method is invoked. For further information, see **"Errors"** on page Intro-20 Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
E_ILLEGAL	It is not recorded.

See Also **StartRecording** Property

Events(UML interfaces)

ErrorEvent *Updated in Release 1.16*

<<event>> **upos::events:: ErrorEvent**

ErrorCode : *int32*{read-write}
ErrorCodeExtended : *int32*{read-write}
ErrorLocus : *int32*{read-write}
*** pErrorResponse : *int32*{read-write}**

Attributes This event contains following attributes.

Attributes	Type	Description
Error Code	*int32*	Error Code causing the error event. See the list of Error Code.
ErrortCodeExtended	*int32*	Error Code causing the error event. These values are device category specific.
ErrorLocus	*int32*	Location of the error. See values below.
pErrorResponse	*int32*	Error response, whose default value may be overridden by the application (i.e., this attribute is settable). See values below.

The ErrorLocus attribute has one of the following values:

Value	Meaning
EL_INPUT	Error occurred while gathering or Processing event-driven input. No previously buffered input data is available.
EL_INPUT_DATA	Error occurred while gathering or processing event-driven input, and some previously buffered data is available.

If ResultCode is E_EXTENDED, ***ResultCodeExtended*** *is set to one of the following values.*

Value	Meaning
ETOT_NOROOM	There is not enough space to create the file.

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

Value	Meaning
ER_CLEAR	I will try its asynchronous output again. The error condition is exited.
ER_CONTINUEINPUT	Only valid when the locus is EL_INPUT_DATA. Acknowledges that a data error has occurred and directs the Device to continue input processing. The Device remains in the error state and will deliver additional DataEvents as directed by the **DataEventEnabled** property. When all input has been delivered and **DataEventEnabled** is again set to true, then another **ErrorEven**t is delivered with locus EL_INPUT. This is the default response when the locus is EL_INPUT_DATA.

Remarks It notifies you when an error is detected during recording.

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

See Also **Status, Error code, State model**

CHAPTER 42

Voice Recognition

This Chapter defines the Voice Recognition device category.

Summary

Properties (UML attributes)

Common	*Type*	*Mutability*	*Version*	*May Use After*
AutoDisable:	*boolean*	{read-write}	1.16	open
CapCompareFirmwareVersion:	*boolean*	{read-only}	1.16	open
CapPowerReporting:	*int32*	{read-only}	1.16	open
CapStatisticsReporting:	*boolean*	{read-only}	1.16	open
CapUpdateFirmware:	*boolean*	{read-only}	1.16	open
CapUpdateStatistics:	*boolean*	{read-only}	1.16	open
CheckHealthText:	*string*	{read-only}	1.16	open
Claimed:	*boolean*	{read-only}	1.16	open
DataCount:	*int32*	{read-only}	1.16	open
DataEventEnabled:	*boolean*	{read-write}	1.16	open
DeviceEnabled:	*boolean*	{read-write}	1.16	open, claim
FreezeEvents:	*boolean*	{read-write}	1.16	open
OutputID:	*int32*	{read-only}	1.16	Not Supported
PowerNotify:	*int32*	{read-write}	1.16	open
PowerState:	*int32*	{read-only}	1.16	open
State:	*int32*	{read-only}	1.16	--
DeviceControlDescription:	*string*	{read-only}	1.16	--
DeviceControlVersion:	*int32*	{read-only}	1.16	--
DeviceServiceDescription:	*string*	{read-only}	1.16	open
DeviceServiceVersion:	*int32*	{read-only}	1.16	open
PhysicalDeviceDescription:	*string*	{read-only}	1.16	open
PhysicalDeviceName:	*string*	{read-only}	1.16	open

Properties (Continued)

Specific	*Type*	*Mutability*	*Version*	*May Use After*
CapLanguage:	*boolean*	{read-only}	1.16	open
HearingDataPattern:	*string*	{read-only}	1.16	open
HearingDataWord:	*string*	{read-only}	1.16	open
HearingDataWordList:	*string*	{read-only}	1.16	open
HearingResult:	*int32*	{read-only}	1.16	open
HearingStatus:	*int32*	{read-only}	1.16	open
LanguageList:	*string*	{read-only}	1.16	open

Methods (UML operations)

Common

Name	*Version*
open (logicalDeviceName: *string*`)`**:** **void {raises-exception}**	1.16
close ( ): **void {raises-exception, use after open}**	1.16
claim (timeout: *int32* **):** **void {raises-exception, use after open}**	1.16
release ( ): **void {raises-exception, use after open, claim}**	1.16
checkHealth (level: *int32* **):** **void {raises-exception, use after open, enable}**	1.16
clearInput ( ): **void { }**	Not supported
clearInputProperties ( ): **void { }**	Not supported
clearOutput ( ): **void { }**	Not supported
compareFirmwareVersion (firmwareFileName: *string,* **out result:** *int32*`)`**:** **void {raises-exception, use after open, claim, enable}**	1.16
directIO (command: *int32,* **inout data:** *int32,* **inout obj:** *object*`)`**:** **void {raises-exception, use after open}**	1.16
resetStatistics (statisticsBuffer: *string*`)`**:** **void {raises-exception, use after open, claim, enable}**	1.16
retrieveStatistics (inout statisticsBuffer: *string*`)`**:** **void {raises-exception, use after open, claim, enable}**	1.16

Methods (UML operations)(continued)

Common

Name	*Version*
updateFirmware (firmwareFileName: *string*`)`**:** `void {raises-exception, use after open, claim, enable}`	1.16
updateStatistics (statisticsBuffer: *string*`)`**:** `void {raises-exception, use after open, claim, enable}`	1.16

Specific

Name	
startHearingFree (language: *string*`)`**:** `void {raises-exception, use after open, claim, enable}`	1.16
startHearingSentence (language: *string*, **wordList:** *string,* **patternList:** *string*`)`**:** `void {raises-exception, use after open, claim, enable}`	1.16
startHearingWord (language: *string*, **wordList:** *string*`)`**:** `void {raises-exception, use after open, claim, enable}`	1.16
startHearingYesNo (language: *string*`)`**:** `void {raises-exception, use after open, claim, enable}`	1.16
stopHearing ( ): `void {raises-exception, use after open, claim, enable}`	1.16

Events (UML interfaces)

Name	*Type*	*Mutability*	*Version*
upos::events::DataEvent			1.16
Status:	*int32*	{read-only}	
upos::events::DirectIOEvent			1.16
EventNumber:	*int32*	{read-only}	
Data:	*int32*	{read-write}	
Obj:	*object*	{read-write}	
upos::events::ErrorEvent			1.16
ErrorCode:	*int32*	{read-only}	
ErrorCodeExtended:	*int32*	{read-only}	
ErrorLocus:	*int32*	{read-only}	
ErrorResponse:	*int32*	{read-write}	
upos::events::OutputCompleteEvent		***Not Supported***	
upos::events::StatusUpdateEvent			1.16
Status:	*int32*	{read-only}	

General Information

The Voice Recognition programmatic name is "VoiceRecognition".

Capabilities

The Voice Recognition has the following capability:

- Convert spoken words to strings.

Voice Recognition Class Diagram

The following diagram shows the relationships between the Voice Recognition classes.

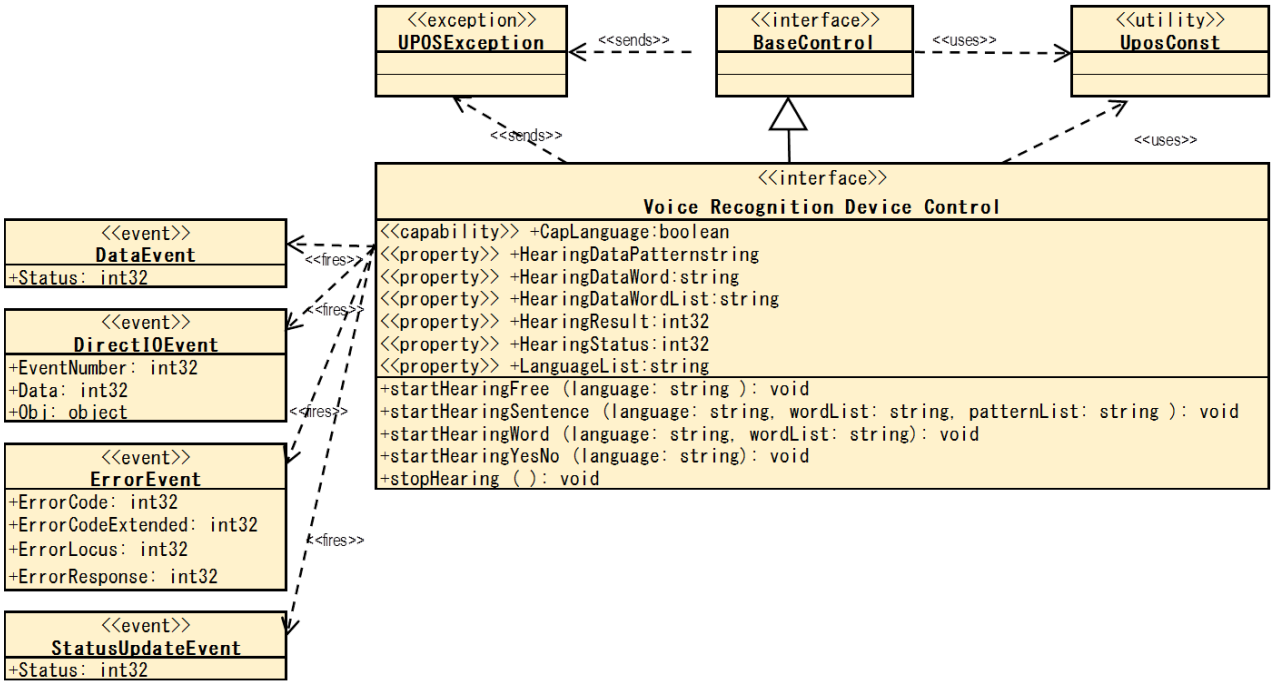


Fig. Chap. 42-1 Voice Recognition Class Diagram

Model

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

Control starts voice recognition with the **startHearingYesNo** method, **startHearingSentence** method, etc., and generates **DataEvent** when recognizing voice.

If the **AutoDisable** property is true, then the device automatically disables itself when a **DataEvent** is enqueued.

An enqueued **DataEvent** can be delivered to the application when the **DataEventEnabled** property is true and other event delivery requirements are met. Just before delivering this event, data is copied into corresponding properties, and further data events are disabled by setting **DataEventEnabled** to false. This causes subsequent input data to be enqueued while the application processes the current input and associated properties. When the application has finished processing the current input and is ready for more data, it reenables events by setting **DataEventEnabled** to true.

An **ErrorEvent** (or events) is enqueued if an error occurs while gathering or processing input, and is delivered to the application when **DataEventEnabled** is true and other event delivery requirements are met.

The **DataCount** property may be read to obtain the total number of enqueued DataEvents.

All enqueued input may be deleted by calling clearInput. See the **clearInput** method description for more details.

All data properties that are populated as a result of firing a **DataEvent** or **ErrorEvent** can be set back to their default values by calling the **clearInputProperties** method.

Types of voice recognition

Voice recognition is mainly a method of specifying word candidates to be recognized and waiting for those words.

There are the following four types of voice recognition.

Yes/No/Cancel recognition

It listens to the sound of words classified as Yes / No / Cancel defined by the device.

For example, the voice ""OK."" is classified as Yes.

The recognized content is set in the HearingDataWord property.

For details, refer to the **startHearingYesNo** method.

Word recognition

The application specifies a list of words and listens for the voice of that word.

The recognized content is set in the HearingDataWord property.

For details, refer to the **startHearingWord** method.

Sentence recognition

The application specifies a word and a list of patterns of the sentences using it and awaits the sound of the sentence.

The recognized content is set in the HearingDataWordList property, **HearingDataPattern** property.

For details, see the **startHearingSentence** method.

Free recognition

Voice recognition leave to the device is performed without specifying the word to wait.

The recognized content is set in the **HearingDataWord** property.

For details, see the **startHearingFree** method.

When recognizing voice, the kind of recognition was stored in the **HearingResult** property.

Device Sharing

The Voice Recognition is an exclusive-use device, as follows:

- The application must claim the device before enabling it.
- The application must claim and enable the device before accessing some properties or calling methods that update the device.
- See the “Summary” table for precise usage prerequisites.

Properties (UML attributes)

CapLanguage Property

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

Remarks If true, the application can change the language. If false, the application cannot change the language.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

HearingDataPattern Property

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

Remarks The pattern ID recognized by the **startHearingSentence** method is set.

This property is set by the control just before the **DataEvent** is notified.

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **startHearingSentence** Method

HearingDataWord Property

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

Remarks The content of voice recognition is set.

This property is set as input data of the following method. To know which method it is for, check the HearingResult property.

Methods	Meaning
startHearingYesNo Method	The recognized word is set.
startHearingWord Method	Recognized words are set among the word candidates specified by the **startHearingWord** method.
startHearingFree Method	Recognized words and sentences are set. The alphabet 's uppercase letters, Japanese kanji, hiragana, katakana, etc., the contents to be set varies depending on the device.

This property is set by the control just before the **DataEvent** is notified.

Errors A **UposException** may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **HearingResult** Property**, startHearingYesNo** Method**,**

startHearingWord Method**, startHearingFree** Method

HearingDataWordList Property

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

Remarks Comma-separated list of word information recognized by the **startHearingSentence** method.

Each word information consists of the following information and is shown in the following order separated by a colon (":").

Parameter	Description
WordGoupiID	Recognized word group ID
Word	Recognized words. The content defined in the word group is set.

For example, in the **startHearingSentence** method, set candidates as follows,
Word list: "Item: coffee: tea, number: one: two"

Sentence pattern: "Pattern 01: [product] as [number], Pattern 02: as [goods] please"

When you recognize the word "one coffee."

In the pattern "Pattern 01", "coffee" of the word group "product" and "one" of "number" are recognized.

At that time, it looks like the following.

"Item: coffee, number: one"

This property is set by the control just before the **DataEvent** is notified.

Errors A **UposException** may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **startHearingSentence** Method

HearingResult Property

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

Remarks A value indicating the voice recognition result is set.

The parameters to be set are as follows.

Value	Meaning
TTS_HRESULT_YESNO_YES	Voice recognition result of Finish running voice recognition. method. Also, Device got an answer that is classified as YES. The recognition content is set in the Finish running voice recognition property.
TTS_HRESULT_YESNO_NO	Voice recognition result of Finish running voice recognition. method. Also, Device got an answer that is classified as NO. The recognition content is set in the **HearingDataWord** property.
TTS_HRESULT_YESNO_CANCEL	Voice recognition result of **startHearingYesNo** method. Also, Device got responses that are classified as CANCEL. The recognition content is set in the **HearingDataWord** property.
TTS_HRESULT_WORD	Recognition result of **startHearingWord** method. The recognition content is set in the **HearingDataWord** property.
TTS_HRESULT_SENTENCE	Recognition result of **startHearingSentence** method. The recognition content is set in the **HearingDataWordList** property, **HearingDataPattern** property.
TTS_HRESULT_FREE	Recognition result of **startHearingFree** method. The recognition content is set in the **HearingDataWord** property.

This property is set by the control just before the **DataEvent** is notified.

Errors A **UposException** may be thrown when this property is accessed.

For further information, see **"Errors"** on page Intro-20.

See Also **HearingDataWord** Property**, HearingDataWordList** Property**, HearingDataPattern** Property**, startHearingYesNo** Method**, startHearingWord** Method**, startHearingSentence** Method**, startHearingFree** Method

HearingStatus Property

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

Remarks A value indicating the voice recognition status is set.

Value	Meaning
TTS_HSTATUS_NONE	Voice recognition is not running.
TTS_HSTATUS_YESNO	Voice recognition by the **startHearingYesNo** method is in progress.
TTS_HSTATUS_WORD	Voice recognition by the **startHearingWord** method is in progress.
TTS_HSTATUS_SENTENCE	Voice recognition by the **startHearingSentence** method is in progress.
TTS_HSTATUS_FREE	Voice recognition by the **startHearingFree** method is in progress.

This property is initialized by the **open** method. Also, it is set by the control just before the voice recognition state changes.

Errors A **UposException** may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **startHearingYesNo** Method**, startHearingWord** Method**, startHearingSentence** Method**, startHearingFree** Method

LanguageList Property

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

Remarks Contains the comma-delimited list of language that are supported by the device. The value representing the language is a value consisting of the language and country code defined in RFC 4664.
For example, when the device supports US / English, Japan / Japanese, it will be as follows.
"en-US, ja-JP"

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **startHearingYesNo** Method**, startHearingWord** Method**, startHearingSentence** Method**, startHearingFree** Method

Methods (UML operations)

startHearingFree Method

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

Parameter	Description
Language	Specify the language to recognize. Specify one of the values listed in the **LanguageList** property.

Remarks Device will start waiting without specifying waiting candidates.

This method is executed asynchronously. You can end voice recognition by calling the **stopHearing** method.

Errors A **UposException** may be thrown when this method is invoked. For further information, see **"Errors"** on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified. Or an unsupported language was specified.
E_BUSY	Voice recognition in progress so it cannot be executed.

See Also **LanguageList** Property**, stopHearing** Method

startHearingSentence Method

Syntax **startHearingSentence (language: *string*, wordList: *string*,**
patternList: *string*):
void {raises-exception, use after open, claim, enable}

Parameter	Description
language	Specify the language to recognize. Specify one of the values listed in the **LanguageList** property.
wordList	Specify word candidates to be waited on in a comma-separated list.
patternList	Specify the sentence pattern information to be waited for in a comma-separated list.

Each word information specified in wordList consists of the following information and is shown in the following order, separated by a colon (":").

Parameter	Description

wordGroupID	ID to identify word list
wordList	A word candidate to be awaited for being separated by a colon (":")

For example, to specify word candidates "one" and "two" for word candidates "coffee" "tea" and word group "number" in the single item group "product", specify as follows.

"Item: coffee: tea, number: one: two"

Each word information specified in patternList consists of the following information, and it is shown in the following order separated by a colon (":").

Parameter	Description
patternID	ID to identify the pattern
pattern	A sentence pattern to wait. To add the word list specified in wordList to the candidate, enclose the word group ID with "[" and "]". Example: "[word group ID 1]" [word group ID 2] "

For example, in wordList, "Item: coffee: tea, number: one: two" is specified, and a pattern requesting goods and number such as "Two coffee please" and a pattern requesting goods such as "Coffee please" When defining, specify as follows.

"Pattern 01: [Number] [Product] Please, Pattern 02: [Product] please"

Remarks Start waiting for sentences defined in wordList and patternList.

This method is executed asynchronously. You can end voice recognition by calling the **stopHearing** method.

Errors A **UposException** may be thrown when this method is invoked. For further information, see **"Errors"** on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified. Or an unsupported language was specified.
E_BUSY	Voice recognition in progress so it cannot be executed.

See Also **LanguageList** Property**, stopHearing** Method

startHearingWord Method

Syntax **startHearingWord (language: *string*, wordList: *string*):**
void {raises-exception, use after open, claim, enable}

Parameter	Description
language	Specify the language to recognize. Specify one of the values listed in the **LanguageList** property.
wordList	Specify word candidates to be waited on in a comma-separated list. Example: "word 1, word 2, word 3"

Remarks Start waiting for word candidates specified in wordList.

This method is executed asynchronously. Application can end voice recognition by calling the **stopHearing** method.

Errors A UposException may be thrown when this method is invoked. For further information, see **"Errors"** on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified. Or an unsupported language was specified.
E_BUSY	Voice recognition in progress so it cannot be executed.

See Also **LanguageList** Property**, stopHearing** Method

startHearingYesNo Method

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

Parameter	Description
language	Specify the language to recognize. Specify one of the values listed in the **LanguageList** property.

Remarks Waiting for word candidates corresponding to "Yes" "No" "Cancel" defined by the device is started.

This method is executed asynchronously. Application can end voice recognition by calling the **stopHearing** method.

Errors A **UposException** may be thrown when this method is invoked. For further information, see **"Errors"** on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified. Or an unsupported language was specified.
E_BUSY	Voice recognition in progress so it cannot be executed.

See Also **LanguageList** Property**, stopHearing** Method

stopHearing Method

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

Remarks Finish running voice recognition.

Errors A **UposException** may be thrown when this method is invoked. For further information, see **"Errors"** on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified. Or an unsupported language was specified.

CHAPTER 43

Sound Player

This Chapter defines the Sound Player device category.

Summary

Properties (UML attributes)

Common	*Type*	*Mutability*	*Version*	*May Use After*
AutoDisable:	*boolean*	{read-write}	1.16	Not Supported
CapCompareFirmwareVersion:	*boolean*	{read-write}	1.16	open
CapPowerReporting:	*int32*	{read-only}	1.16	open
CapStatisticsReporting:	*boolean*	{read-only}	1.16	open
CapUpdateFirmware:	*boolean*	{read-only}	1.16	open
CapUpdateStatistics:	*boolean*	{read-only}	1.16	open
CheckHealthText:	*string*	{read-only}	1.16	open
Claimed:	*boolean*	{read-only}	1.16	open
DataCount:	*int32*	{read-only}	1.16	open
DataEventEnabled:	*boolean*	{read-write}	1.16	open
DeviceEnabled:	*boolean*	{read-write}	1.16	open, claim
FreezeEvents:	*boolean*	{read-write}	1.16	open
OutputID:	*nt32*	{read-only}	1.16	open
PowerNotify:	*nt32*	{read-write}	1.16	open
PowerState:	*nt32*	{read-only}	1.16	open
State:	*nt32*	{read-only}	1.16	--
DeviceControlDescription:	*string*	{read-only}	1.16	-
DeviceControlVersion:	*int32*	{read-only}	1.16	-
DeviceServiceDescription:	*string*	{read-only}	1.16	open
DeviceServiceVersion:	*Int32*	{read-only}	1.16	open
PhysicalDeviceDescription:	*string*	{read-only}	1.16	open
PhysicalDeviceName:	*string*	{read-only}	1.16	open

Properties (Continued)

Specific	*Type*	*Mutability*	*Version*	*May Use After*
CapVolume:	*boolean*	{read-only}	1.16	open
CapMultiPlay:	*boolean*	{read-only}	1.16	open
CapSoundTypeList:	*string*	{read-only}	1.16	open
DeviceSoundList:	*string*	{read-only}	1.16	open
Volume:	*int32*	{read-write}	1.16	open, claim & enable
OutputIDList:	*string*	{read-only}	1.16	open, claim & enable

Methods (UML operations)

Common

Name	*Version*
open (logicalDeviceName: *string*`)`**:** **void {raises-exception}**	1.16
close ( ): **void {raises-exception, use after open}**	1.16
claim (timeout: *int32*`)`**:** **void {raises-exception, use after open}**	1.16
release ( ): **void {raises-exception, use after open, claim}**	1.16
checkHealth (level: *int32*`)`**:** **void {raises-exception, use after open, enable}**	1.16
clearInput ( ): **void { }**	***Not supported***
clearInputProperties ( ): **void { }**	***Not supported***
clearOutput ( ): **void { }**	***Not supported***
directIO (command: *int32*`,` **inout data:** *int32*`,` **inout obj:** *object*`)`**:** **void {raises-exception, use after open}**	1.16
compareFirmwareVersion (firmwareFileName: *string*`,` **out result:** *int32*`)`**:** **void {raises-exception, use after open, claim, enable}**	1.16
resetStatistics (statisticsBuffer: *string*`)`**:** **void {raises-exception, use after open, claim, enable}**	1.16
retrieveStatistics (inout statisticsBuffer: *string* **):** **void {raises-exception, use after open, claim, enable}**	1.16

Methods (UML operations)(continued)

Common

Name	*Version*
updateFirmware (firmwareFileName: *string***):** **void {raises-exception, use after open, claim, enable}**	1.16
updateStatistics (statisticsBuffer: *string***):** **void {raises-exception, use after open, claim, enable}**	1.16

Specific

Name	*Version*
playSound(fileName: *string***, loop:** *boolean***):** **void { raises-exception, use after open, claim, enable}**	1.16
stopSound(outputID:*int32***):** **void {raises-exception, use after open, claim, enable}**	1.16

Events (UML interfaces)

Name	*Type*	*Mutability*	*Version*
upos::events::DataEvent		***Not Supported***	1.16
upos::events::DirectIOEvent			1.16
EventNumber:	*int32*	{read-only}	
Data:	*int32*	{read-write}	
Obj:	*object*	{read-write}	
upos::events::ErrorEvent			1.16
ErrorCode:	*int32*	{read-only}	
ErrorCodeExtended:	*int32*	{read-only}	
ErrorLocus:	*int32*	{read-only}	
ErrorResponse:	*int32*	{read-write}	
upos::events::OutputCompleteEvent			1.16
OutputID:	*int32*	{read-only}	
upos::events::StatusUpdateEvent			1.16
Status:	*int32*	{read-only}	

General Information

The Sound Player programmatic name is "SoundPlayer".

Capabilities

The Sound Player has the following capability:

- Play audio file.

Sound Player Class Diagram

The following diagram shows the relationships between the Sound player classes.

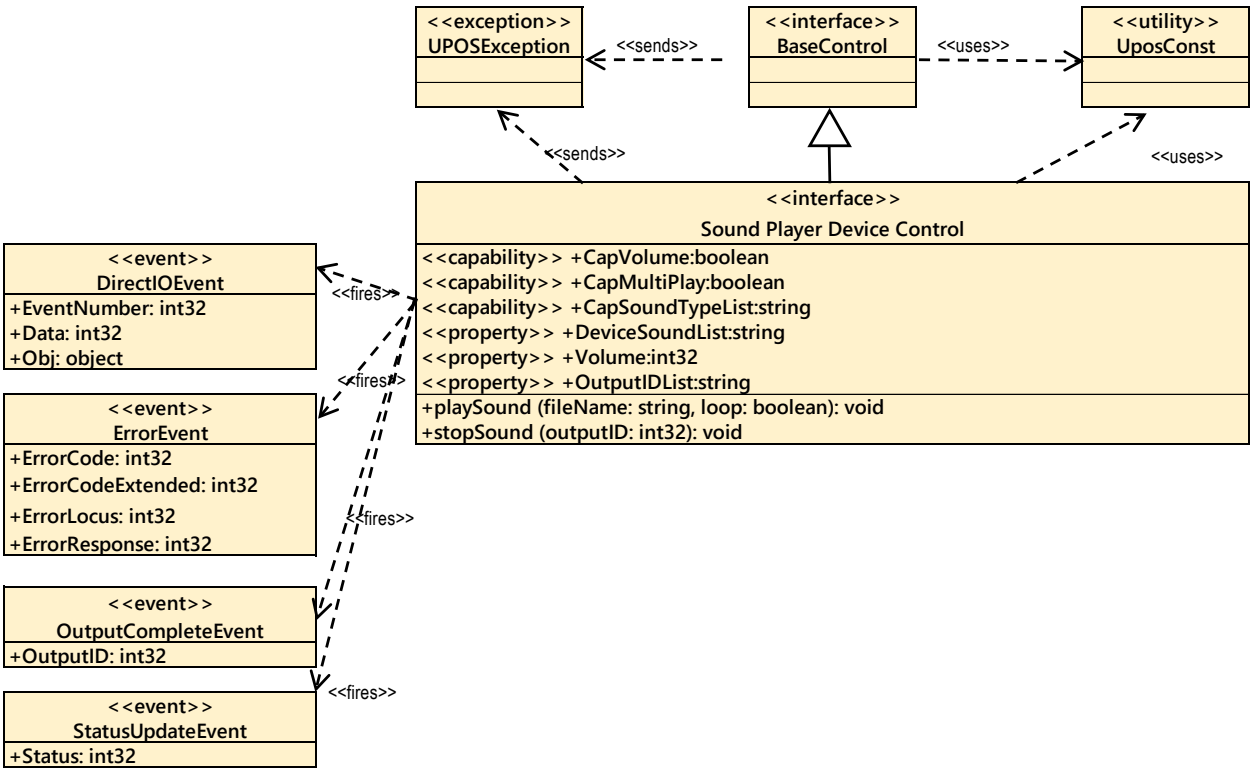


Fig. Chap.43-1 Sound Playter Class Diagram

Model

The Sound Player follows the general device behavior model for asynchronous output devices:

- The application calls a **startSound** method to start playing sound. The Device validates the method parameters and produces an error condition immediately if necessary. If the validation is successful, the Device does the following:
- "1. Buffers the request in program memory, for delivery to the Physical Device as soon as the Physical Device can receive and process it.
- 2. Sets the OutputID property to a unique integer identifier for this request.
- 3. Returns as soon as possible."
- When the Device successfully completes a request, an **OutputCompleteEvent** is enqueued for delivery to the application. A property of this event contains the output ID of the completed request. The application should compare the returned **OutputCompleteEvent** property **OutputID** value with the **OutputID** value set by the asynchronous process method call used to send the data in order to track what data has been successfully sent to the device.
- If an error occurs while processing a request, an ErrorEvent is enqueued which will be delivered to the application after the events already enqueued, including **OutputCompleteEvent**s .I No further asynchronous output will occur until the event has been delivered to the application. If the response is ER_CLEAR, then outstanding asynchronous output is cleared. If the response is ER_RETRY, then output is retried; note that if several outputs were simultaneously in progress at the time that the error was detected, then the Service may need to retry all of these outputs.
- Asynchronous output is always performed on a first-in first-out basis. If the device supports concurrent playback, the request will be executed simultaneously. To check if the device supports simultaneous playback, check the **CapMultiPlay** property.
- "If the request is terminated before completion, due to reasons such as the application calling the **clearOutput** method, then no **OutputCompleteEvent** is delivered.
- Application can also delete the output individually by calling the **stopSound** method. Also in this case **OutputCompleteEvent** will not be notified."
- The **CapSoundTypeList** property lists audio files that the device can play.
- Applications need to support "hard total" services as audio files played with the **startSound** method must be placed in the area managed by the "hard total" service.

Device Sharing

The Sound Player is an exclusive-use device, as follows:

- The application must claim the device before enabling it.
- The application must claim and enable the device before accessing some properties or calling methods that update the device.
- See the “Summary” table for precise usage prerequisites.

Properties(UML attributes)

CapVolume Property

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

Remarks If true, the application can change the volume during playback.

If false, the application cannot change the volume during playback.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **Volume** Property.

CapMultiPlay Property

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

Remarks If true, the application can play sound simultaneously.

If false, the application cannot play sound simultaneously.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **playSound** Method.

CapSoundTypeList Property

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

Remarks Contains the comma-delimited list of file type that is supported by the device.

For example, if the device only supports WAV and OGG, then this property should be set to "WAV,OGG". This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **playSound** Method

DeviceSoundList Property

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

Remarks Contains the comma-delimited list of device sound ID that is supported by the device. This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **playSound** Method

OutputIDList Property

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

Remarks Contains the comma-delimited list of OutputID that is output by the **playSound** method. This property is initialized by the **open** method.

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **playSound** Method

Volume Property

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

Remarks Holds the volume at playing sound.

Legal values range from zero through 100.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

Value	Meaning
E_ILLEGAL	An invalid value was specified.

See Also **playSound** Method

Methods (UML operations)

playSound Method

Syntax **playSound (fileName : *string*, loop : *boolean*):**
void{raises-exception, use after open, claim, enable}

Parameter	Description
fileName	Specifies the file name of audio file. Or, Specifies one of the sound ID defined by **DeviceSoundList**.
loop	When true is specified, loop playback is performed, and if false is specified, loop playback will not be performed.

Remarks Play audio file specified by fileName or device definition sound.

Audio files must be located in the area managed by "Hard Total" service.

This method will be performed asynchronously. To stop playback, call the **stopSound** method.

Errors A UposException may be thrown when this method is invoked. For further information , see “Errors” on page Intro-20. Some possible values of the exception’s ErrorCode property are:

Value	Meaning
E_ILLEGAL	An invalid value was specified. Or an unsupported sound file was specified.
E_NOEXIST	File does not exist.

See Also **CapSoundType** Property, **DeviceSoundList** Property, **stopSound** Method

stopSound Method

Syntax **StopSound(outputID: *int32*):**

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

Parameter	Description
outputID	Specify the outputID of the sound to stop.

Remarks Terminates specified audio playback.

Errors A UposException may be thrown when this method is invoked. For further information, see “Errors” on page Intro-20. Some possible values of the exception’s ErrorCode property are:

Value	Meaning
E_ILLEGAL	The specified sound is not being played.

See Also **OutputID** Property, **startSound** Method

CHAPTER 44

Speech Synthesis

This Chapter defines the Speech Synthesis device category.

Summary

Properties (UML attributes)

Common	*Type*	*Mutability*	*Version*	*May Use After*
AutoDisable:	*boolean*	{read-write}	1.16	Not Supported
CapCompareFirmwareVersion:	*boolean*	{read-only}	1.16	open
CapPowerReporting:	*int32*	{read-only}	1.16	open
CapStatisticsReporting:	*boolean*	{read-only}	1.16	open
CapUpdateFirmware:	*boolean*	{read-only}	1.16	open
CapUpdateStatistics:	*boolean*	{read-only}	1.16	open
CheckHealthText:	*string*	{read-only}	1.16	open
Claimed:	*boolean*	{read-only}	1.16	open
DataCount:	*int32*	{read-only}	1.16	Not Supported
DataEventEnabled:	*boolean*	{read-write}	1.16	Not Supported
DeviceEnabled:	*boolean*	{read-write}	1.16	open, claim
FreezeEvents:	*boolean*	{read-write}	1.16	open
OutputID:	*int32*	{read-only}	1.16	open
PowerNotify:	*int32*	{read-write}	1.16	open
PowerState:	*int32*	{read-only}	1.16	open
State:	*int32*	{read-only}	1.16	--
DeviceControlDescription:	*string*	{read-only}	1.16	--
DeviceControlVersion:	*int32*	{read-only}	1.16	--
DeviceServiceDescription:	*string*	{read-only}	1.16	open
DeviceServiceVersion:	*int32*	{read-only}	1.16	open
PhysicalDeviceDescription:	*string*	{read-only}	1.16	open
PhysicalDeviceName:	*string*	{read-only}	1.16	open

Properties (Continued)

Specific	*Type*	*Mutability*	*Version*	*May Use After*
CapLanguage:	*boolean*	{read-only}	1.16	open
CapPitch:	*boolean*	{read-only}	1.16	open
CapSpeed:	*boolean*	{read-only}	1.16	open
CapVoice:	*boolean*	{read-only}	1.16	open
CapVolume:	*boolean*	{read-only}	1.16	open
Language:	*string*	{read-write}	1.16	open, claim & enable
LanguageList:	*string*	{read-only}	1.16	open
OutputIDList:	*string*	{read-only}	1.16	open, claim & enable
Pitch:	*int32*	{read-write}	1.16	open, claim & enable
Speed:	*int32*	{read-write}	1.16	open, claim & enable
Voice:	*string*	{read-write}	1.16	open, claim & enable
VoiceList:	*string*	{read-only}	1.16	open
Volume:	*int32*	{read-write}	1.16	open, claim & enable

Methods (UML operations)

Common

Name	*Version*
open (logicalDeviceName: *string***):** **void {raises-exception}**	1.16
close ( ): **void {raises-exception, use after open}**	1.16
claim (timeout: *int32***):** **void {raises-exception, use after open}**	1.16
release ( ): **void {raises-exception, use after open, claim}**	1.16
checkHealth (level: *int32***):** **void {raises-exception, use after open, enable}**	1.16
clearInput ( ): **void { }**	Not supported
clearInputProperties ( ): **void { }**	Not supported

Methods (UML operations)(continued)

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

Common

Name	*Version*
compareFirmwareVersion (firmwareFileName: *string,* **out result:** *int32***):** **void {raises-exception, use after open, claim, enable}**	1.16
directIO (command: *int32,* **inout data:** *int32,* **inout obj:** *object***):** **void {raises-exception, use after open}**	1.16
resetStatistics (statisticsBuffer: *string***):** **void {raises-exception, use after open, claim, enable}**	1.16
retrieveStatistics (inout statisticsBuffer: *string***):** **void {raises-exception, use after open, claim, enable}**	1.16
updateFirmware (firmwareFileName: *string***):** **void {raises-exception, use after open, claim, enable}**	1.16
updateStatistics (statisticsBuffer: *string***):** **void {raises-exception, use after open, claim, enable}**	1.16

Specific

Name	
speak (text: *string***):** **void {raises-exception, use after open, claim, enable}**	1.16
speakImmediate (text: *string***):** **void {raises-exception, use after open, claim, enable}**	1.16
stopCurrentSpeaking ( ): **void {raises-exception, use after open, claim, enable}**	1.16
stopSpeaking (outputID: *int32* **):** **void {raises-exception, use after open, claim, enable}**	1.16

Events (UML interfaces)

Name	*Type*	*Mutability*	*Version*
upos::events::DataEvent		***Not Supported***	
upos::events::DirectIOEvent			1.16
EventNumber:	*int32*	{read-only}	
Data:	*int32*	{read-write}	
Obj:	*object*	{read-write}	
upos::events::ErrorEvent			1.16
ErrorCode:	*int32*	{read-only}	
ErrorCodeExtended:	*int32*	{read-only}	
ErrorLocus:	*int32*	{read-only}	
ErrorResponse:	*int32*	{read-write}	
upos::events::OutputCompleteEvent			1.16
OutputID:	*int32*	{read-only}	
upos::events::StatusUpdateEvent			1.16
Status:	*int32*	{read-only}	

General Information

The Speech Synthesis programmatic name is "SpeechSynthesis".

Capabilities

The Speech Synthesis has the following capability:

- Convert text to speech and speak.

Speech Synthesis Class Diagram

The following diagram shows the relationships between the Speech Synthesis classes.

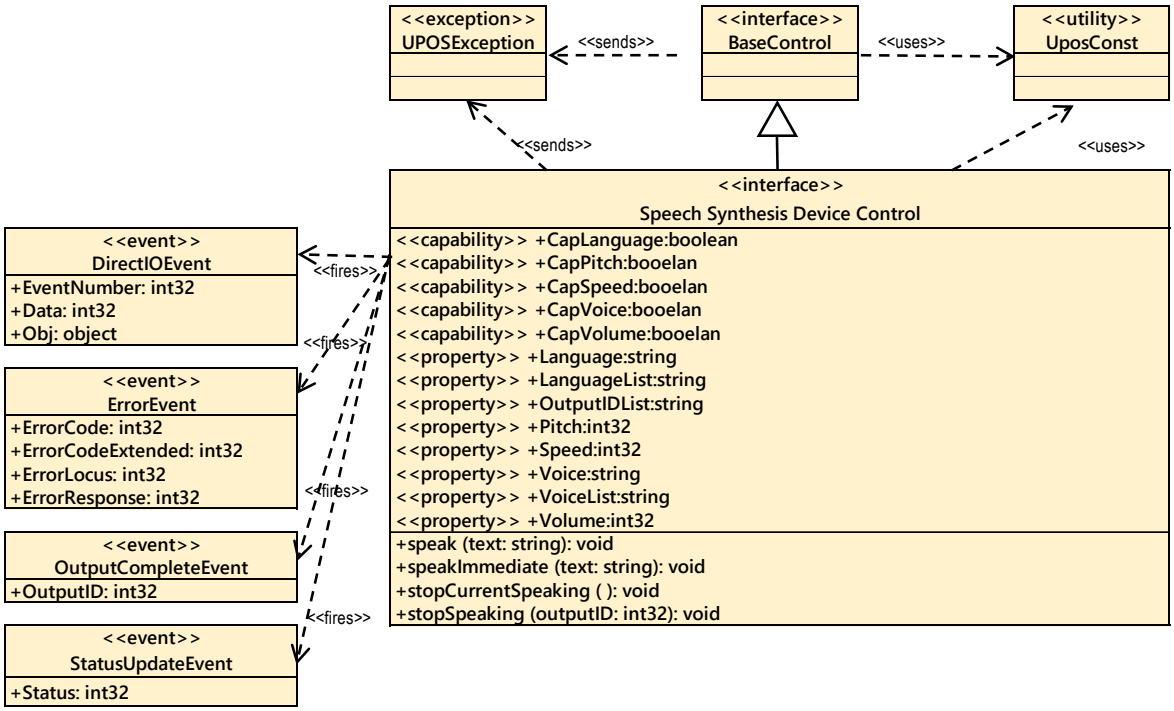


Fig. Chap. 44-1 Speech Synthesis Class Diagram

Model

The Speech Synthesis follows the general device behavior model for asynchronous output devices:

The application calls a **speak** method or **speakImmediate** method to speech.

The speak method acts to start speaking the words specified by text, while the speakImmediate method ends immediately previous speak method, and starts speaking the word specified by text asynchronously and immediately.

The device validates the method parameters and produces an error condition immediately if necessary. If the validation is successful, the device does the following:

1. Buffers the request in program memory, for delivery to the physical device as soon as the physical device can receive and process it.
2. Sets the **OutputID** property to a unique integer identifier for this request.
3. Returns as soon as possible.

When the device successfully completes a request, an **OutputCompleteEvent** is enqueued for delivery to the application. A property of this event contains the output ID of the completed request. The application should compare the returned **OutputCompleteEvent** property's **OutputID** value with the OutputID value set by the asynchronous process method call used to send the data in order to track what data has been successfully sent to the device.

If an error occurs while processing a request, an **ErrorEvent** is enqueued which will be delivered to the application after the events already enqueued, including **OutputCompleteEvent**. No further asynchronous output will occur until the event has been delivered to the application. If the response is ER_CLEAR, then outstanding asynchronous output is cleared. If the response is ER_RETRY, then output is retried; note that if several outputs were simultaneously in progress at the time that the error was detected, then the service may need to retry all of these outputs.

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

If the request is terminated before completion, due to reasons such as the application calling the **clearOutput** method, then no **OutputCompleteEvent** is delivered.

Application can also delete the output individually by calling the **stopCurrentSpeaking**, **stopSpeaking** method. Also in this case **OutputCompleteEvent** will not be notified.

Device Sharing

The Speech Synthesis is an exclusive-use device, as follows:

- The application must claim the device before enabling it.
- The application must claim and enable the device before accessing some properties or calling methods that update the device.
- See the “Summary” table for precise usage prerequisites.

Properties (UML attributes)

CapLanguage Property

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

Remarks If true, the application can change the language. If false, the application cannot change the language.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **Language** Property

CapPitch Property

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

Remarks If true, the application can change the pitch. If false, the application cannot change the pitch.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **Pitch** Property

CapSpeed Property

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

Remarks If true, the application can change the speed. If false, the application cannot change the speed.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **Speed** Property

CapVoice Property

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

Remarks If true, the application can change the voice. If false, the application cannot change the voice.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **Voice** Property

CapVolume Property

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

Remarks If true, the application can change the volume. If false, the application cannot change the volume.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **Volume** Property

Language Property

Syntax **Language:** ***string*** **{read-write, access after open, claim, enable}**

Remarks Indicates the language to speak. Valid values are one of the values listed in the **LanguageList** property.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified. Or an unsupported language was specified.

See Also **speak** Method, **speakImmediate** Method

LanguageList Property

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

Remarks Contains the comma-delimited list of language that are supported by the device. The value representing the language is a value consisting of the language and country code defined in RFC 4664. For example, when the device supports US / English, Japan / Japanese, it will be as follows.
"en-US, ja-JP"

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **Language** Property

OutputIDList Property

Syntax **OutputIDList :** ***string*** **{read-write, access after open, claim, enable}**

Remarks Comma-separated list of **OutputID** property values of audio being played by **Speak** method or **SpeakImmediate** method.

This property is initialized by the **open** method. It will also be updated as the speech request increases or decreases.

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **speak** Method, **speakImmediate** Method

Pitch Property

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

Remarks Holds the pitch at speech. Legal values range from 50% through 200%.

This property is initialized to 100% by the **open** method.

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified.

See Also **speak** Method, **speakImmediate** Method

Speed Property

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

Remarks Holds the speed at speech. Legal values range from 50% through 200%.

This property is initialized to 100% by the **open** method.

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified.

See Also **speak** Method, **speakImmediate** Method

Voice Property

Syntax **Voice :** ***string*** **{read-write, access after open, claim, enable }**

Remarks Indicates the voice tone to speak. Valid values are one of the values listed in the **VoiceList** property.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified. Or an unsupported voice was specified.

See Also **speak** Method, **speakImmediate** Method

VoiceList Property

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

Remarks A list of speech able voices are shown in a comma-separated list. For example, when the device supports male and female voice tones, it looks like the following.
"MALE_VOICE, FEMALE_VOICE"
(The content of the value depends on the device)

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **Voice** Property

Volume Property

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

Remarks Holds the volume at speech. Legal values range from zero through 100.

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified.

See Also **speak** Method, **speakImmediate** Method

Methods (UML operations)

speak Method

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

Parameter	Description
text	Specify the text to speak.

Remarks Device will utter the words specified by Text.

The utterance is executed according to the setting contents of **Language** property, **Volume** property, **Pitch** property, **Speed** property, but by inserting the following tag in the text, it is possible to change the utterance after the tag.

Tag	Description
volume	Specify the volume of the uttered voice. Valid values range from 1 to 100.
pitch	Specify the high or low of the uttered voice. Valid values range from 50 to 200.
speed	Specify the speed of the uttered voice. Valid values range from 50 to 200.
pause	Specify the time to pause in milliseconds.
reset	Delete the effect of volume, pitch, speed.

Tags without reset are specified in the form of "\\ tag = value \\". For example, when specifying Text as follows, "Hello \\ pause = 1000 \ \ \ pitch = 150 \ \ It's nice weather today \ \ reset \\". "Hello" speaks according to the

original setting. Then wait for 1000 milliseconds. "Today" speaks Pitch at 150%. "Nice weather," I will speak according to the original settings.

If the device does not support Volume change etc, that tag will be ignored.

This method is executed asynchronously. To end an utterance halfway, call the **stopCurrentSpeaking** method or the **stopSpeaking** method.

Errors A UposException may be thrown when this method is invoked. For further information, see **"Errors"** on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified. The language set in the **Language** property and the language specified by Text do not match.

See Also **Language** Property**, Volume** Property, **Pitch** Property, **Speed** Property, **stopCurrentSpeaking** Method, **stopSpeaking** Method

speakImmediate Method

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

Parameter	Description
text	Specify the text to speak.

Remarks The speak method acts to start speaking the words specified by text, while the speakImmediate method ends immediately previous speak method, and starts speaking the word specified by text asynchronously and immediately.

After executing the same processing as the **clearOutput** method, speak the wording specified by text.

Like this **speak** method, this method can also change a specific wording by inserting a tag. For details, refer to the description of **speak** method.

This method is executed asynchronously. To end an utterance halfway, call the **stopCurrentSpeaking** method or the **stopSpeaking** method.

Errors A UposException may be thrown when this method is invoked. For further information, see **"Errors"** on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified. The language set in the **Language** property and the language specified by Text do not match.

See Also **Language** Property**, Volume** Property, **Pitch** Property, **Speed** Property, **stopCurrentSpeaking** Method, **stopSpeaking** Method

stopCurrentSpeaking Method

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

Remarks Stops the currently executed utterance.

Errors A UposException may be thrown when this method is invoked. For further information, see **"Errors"** on page Intro-20.

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

Value	Meaning
E_ILLEGAL	Speech is not running.

See Also **speak** Method, **speakImmediate** Method

stopSpeaking Method

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

Parameter	Description
outputID	Specify the value of the **OutputID** property you wish to terminate.

Remarks Stop and delete the utterance specified in OutputID.

Errors A UposException may be thrown when this method is invoked. For further information, see **"Errors"** on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified.

See Also **OutputID** Property, **speak** Method, **speakImmediate** Method

CHAPTER 45

Gesture Control

This Chapter defines the Gesture Control device category.

Summary

Properties (UML attributes)

Common	*Type*	*Mutability*	*Version*	*May Use After*
AutoDisable:	*boolean*	{read-write}	1.16	open
CapCompareFirmwareVersion:	*boolean*	{read-only}	1.16	open
CapPowerReporting:	*int32*	{read-only}	1.16	open
CapStatisticsReporting:	*boolean*	{read-only}	1.16	open
CapUpdateFirmware:	*boolean*	{read-only}	1.16	open
CapUpdateStatistics:	*boolean*	{read-only}	1.16	open
CheckHealthText:	*string*	{read-only}	1.16	open
Claimed:	*boolean*	{read-only}	1.16	open
DataCount:	*int32*	{read-only}	1.16	open
DataEventEnabled:	*boolean*	{read-write}	1.16	open
DeviceEnabled:	*boolean*	{read-write}	1.16	open, claim
FreezeEvents:	*boolean*	{read-write}	1.16	open
OutputID:	*int32*	{read-only}	1.16	open
PowerNotify:	*int32*	{read-write}	1.16	open
PowerState:	*int32*	{read-only}	1.16	open
State:	*int32*	{read-only}	1.16	--
DeviceControlDescription:	*string*	{read-only}	1.16	--
DeviceControlVersion:	*int32*	{read-only}	1.16	--
DeviceServiceDescription:	*string*	{read-only}	1.16	open
DeviceServiceVersion:	*int32*	{read-only}	1.16	open
PhysicalDeviceDescription:	*string*	{read-only}	1.16	open

Properties (Continued)

Specific	*Type*	*Mutability*	*Version*	*May Use After*
JointList:	*string*	{read-only}	1.16	open
AutoModeList:	*string*	{read-only}	1.16	open
AutoMode:	*string*	{read-write}	1.16	open, claim & enable
CapMotion:	*boolean*	{read-only}	1.16	open
CapPose:	*boolean*	{read-only}	1.16	open
CapMotionCreation:	*boolean*	{read-only}	1.16	open
CapPoseCreation:	*boolean*	{read-only}	1.16	open
MotionList:	*string*	{read-only}	1.16	open
PoseList:	*string*	{read-only}	1.16	open
PoseCreationMode:	*boolean*	{read-write}	1.16	open, claim & enable
PhysicalDeviceName:	*string*	{read-only}	1.16	open

Methods (UML operations)

Common

Name	*Version*
open (logicalDeviceName: *string***):** **void {raises-exception}**	1.16
close ( ): **void {raises-exception, use after open}**	1.16
claim (timeout: *int32***):** **void {raises-exception, use after open}**	1.16
release ( ): **void {raises-exception, use after open, claim}**	1.16
checkHealth (level: *int32***):** **void {raises-exception, use after open, enable}**	1.16
clearInput ( ): **void { }**	Not supported
clearInputProperties ( ): **void { }**	Not supported
clearOutput ( ): **void { }**	Not supported
compareFirmwareVersion (firmwareFileName: *string*, **out result:** *int32***):** **void {raises-exception, use after open, enable}**	1.16
directIO (command: *int32*, **inout data:** *int32*, **inout obj:** *object***):** **void {raises-exception, use after open}**	1.16
resetStatistics (statisticsBuffer: *string***):** **void {raises-exception, use after open, enable}**	1.16
retrieveStatistics (inout statisticsBuffer: *string***):** **void {raises-exception, use after open, enable}**	1.16
updateFirmware (firmwareFileName: *string***):** **void {raises-exception, use after open, enable}**	1.16
updateStatistics (statisticsBuffer: *string***):** **void {raises-exception, use after open, enable}**	1.16

Methods (UML operations)(continued)

Specific

Name	*Version*
setPotision (positionList: *string,* **time:** *int32 ,* **absolute:** *boolean*)**:** **void { raises-exception, use after open, claim, enable }**	1.16
setSpeed (speedList: *string*, **time:** *int32*)**:** **void { raises-exception, use after open, claim, enable }**	1.16
getPosition (jointID: *string*, **position:** *int32 by reference*)**:** **void { raises-exception, use after open, claim, enable }**	1.16
startMotion (fileName: *string*)**:** **void { raises-exception, use after open, claim, enable }**	1.16
createMotion (fileName: *string*, **poseList:** *string*)**:** **void { raises-exception, use after open, claim, enable }**	1.16
startPose (fileName: *string*)**:** **void { raises-exception, use after open, claim, enable }**	1.16
createPose (fileName: *string*, **time:** *int32*)**:** **void { raises-exception, use after open, claim, enable }**	1.16
stopControl (outputID: *int32*)**:** **void { raises-exception, use after open, claim, enable }**	1.16

Events (UML interfaces)

Name	*Type*	*Mutability*	*Version*
upos::events::DataEvent		***Not Supported***	
upos::events::DirectIOEvent			1.16
EventNumber:	*int32*	{read-only}	
Data:	*int32*	{read-write}	
Obj:	*object*	{read-write}	
upos::events::ErrorEvent			1.16
ErrorCode:	*int32*	{read-only}	
ErrorCodeExtended:	*int32*	{read-only}	
ErrorLocus:	*int32*	{read-only}	
ErrorResponse:	*int32*	{read-write}	
upos::events::OutputCompleteEvent			1.16
OutputID:	*int32*	{read-only}	
upos::events::StatusUpdateEvent			1.16
Status:	*int32*	{read-only}	

General Information

The Gesture Control programmatic name is "GestureControl".

Capabilities

The Gesture Control has the following capability:

- It controls the operation of various joints.
- The operation is automatically controlled by interlocking various joints and other devices.
- Register and play the defined pose and motion.

Gesture Control Class Diagram

The following diagram shows the relationships between the Gesture Control classes.

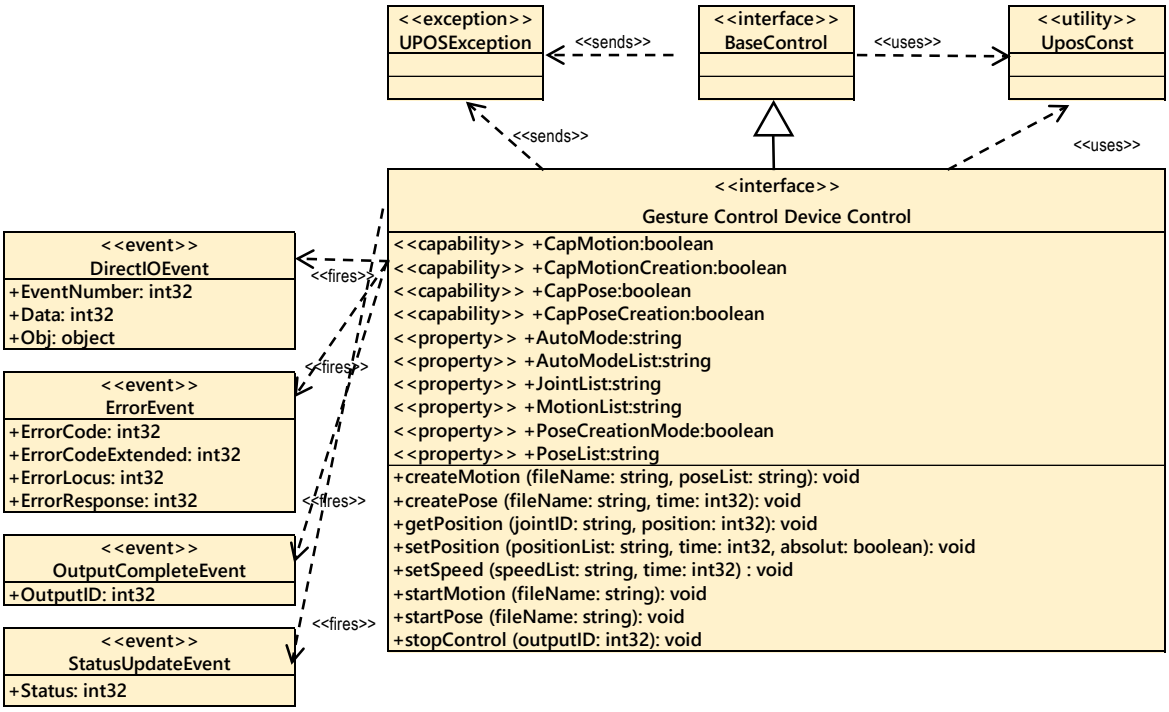


Fig. Chap. 45-1 Gesture Control Class Diagram

Model

The Gesture Control follows the general device behavior model for asynchronous output devices:

- The application calls a **setPosition**, s**etSpeed**, s**tartPose**, s**tartMotion** method to start output. The Device validates the method parameters and produces an error condition immediately if necessary. If the validation is successful, the Device does the following:
  1. Buffers the request in program memory, for delivery to the Physical Device as soon as the Physical Device can receive and process it.
  2. Sets the **OutputID** property to a unique integer identifier for this request.
  3. Returns as soon as possible.
- When the Device successfully completes a request, an **OutputCompleteEvent** is enqueued for delivery to the application. A property of this event contains the output ID of the completed request. The application should compare the returned OutputCompleteEvent property OutputID value with the OutputID value set by the asynchronous process method call used to send the data in order to track what data has been successfully sent to the device.
- If an error occurs while processing a request, an **ErrorEvent** is enqueued which will be delivered to the application after the events already enqueued, including **OutputCompleteEvent**. No further asynchronous output will occur until the event has been delivered to the application. If the response is ER_CLEAR, then outstanding asynchronous output is cleared. If the response is ER_RETRY, then output is retried; note that if several outputs were simultaneously in progress at the time that the error was detected, then the Service may need to retry all of these outputs.
- Asynchronous output is always performed on a first-in first-out basis.
- If the request is terminated before completion, due to reasons such as the application calling the **clearOutput** method, then no **OutputCompleteEvent** is delivered.
- Application can also delete the output individually by calling the **stopControl** method. Also in this case **OutputCompleteEvent** will not be notified.

Automatic control

Automatic control of a joint means to automatically control a joint on the device side, such as tracking according to the movement of a person's face, in cooperation with a camera or the like connected to the device.

The automatic control function is device dependent. For possible automatic control, it is enabled by confirming with the **AutoModeList** property and setting a value in the **AutoMode** property.

Pose / Motion

Pose refers to setting the position of one or more defined joints.

For example, it is an action that lifts a hand.

To execute a pose, specify the pose file name in the **startPose** method or the pose name defined in the device.

Create the pose file with the **createPose** method described later. Pose defined on the device will check the **PoseList** property.

To execute motion, specify the motion file name or the motion name defined in the device in the **startMotion** method.

Motion files are created by the **createMotion** method to be described later. Motion defined on the device will check the **MotionList** property.

To create a pose file, first set the **PoseCreationMode** property to TRUE and enable the pose registration function. When pose registration function is enabled, each joint is set to the default position. At this time, if the automatic control mode is enabled, the automatic control mode is temporarily invalidated.

Application can then create a pose file by setting the value you want to define as a pose with the **setPosition** method and calling the **createPose** method.

A motion file can be created by specifying the pose defined by the created pause file or device and calling the **createMotion** method.

Since the created pause and motion files are recorded in the area managed by the "hard total" service, the application must also support "hard total" service.

Device Sharing

The Gesture Control is an exclusive-use device, as follows:

- The application must claim the device before enabling it.
- The application must claim and enable the device before accessing some properties or calling methods that update the device.
- See the “Summary” table for precise usage prerequisites.

Properties (UML attributes)

JointList Property

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

Remarks Comma-separated list of joint information supported by the device.

Each piece of joint information consists of the following information and is shown in the following order, separated by a colon (":").

Parameter	Description
JointID	Indicates a unique ID in the service that identifies the joint. Position range availability If 0, the joint does not have the position range, 1 holds the position range. For example, the arm joint has a range of rotation width, but the wheel for movement does not have the range of movement amount. For example, for a device that supports pitch, roll, and yaw joints and a device that supports rotation by wheel and joint that can move forward and backward, it is as follows. "Joint 01 _Pitch: 1, Joint 01 _ Roll: 1, Joint 01 _ Yaw: 1, Wheel_Turn: 0, Wheel_Move: 0" This property is initialized by the **open** method.

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

AutoModeList Property

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

Remarks Comma-separated list of joint automatic control IDs supported by the device.

For example, in conjunction with the camera, if the mode of tracking the face of a person by moving only the joint of Joint 01 and the mode of tracking by moving all joints are supported as follows.

"FaceTrack_Joint 01, FaceTrack_ALL"

(Content and order are dependent on the device.)

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

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

See Also **AutoMode** Property.

AutoMode Property

Syntax **AutoMode: *string* {read-write, access after open, claim, enable}**

Remarks Indicates automatic control mode ID. Valid values are the empty string "" or one of the **AutoModeList** properties listed.

If you set one of the properties described in the **AutoModeList** property for this property, the automatic control mode will be enabled in the set mode.

Setting the empty character "" disables the automatic control mode.

This property is initialized to the empty string "" by the **open** method.

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified.

See Also **AutoModeList** Property

CapMotion Property

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

Remarks If true, the device supports pose function.

If false, the device does not support pose function.

If this property is false, change of **PoseCreationMode** property, **startPose** method, **createPose** method is not available.

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

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

See Also **startMotion** Method, **createMotion** Method.

CapPose Property

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

Remarks If true, the device supports pose function.

If false, the device does not support pose function.

If this property is FALSE, change of **PoseCreationMode** property, **startPose** method, **createPose** method is not available.

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

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

See Also **PoseCreationMode** Property, **startPose** Method, **createPose** Method.

CapMotionCreation Property

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

Remarks If true, the device supports motion registration function.

If false, the device does not support motion registration function.

If this property is FALSE, the **createMotion** method is not available.

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

Errors A UposException may be thrown when this method is invoked. For further information, see “**Errors**" on page Intro-20.

See Also **createMotion** Method.

CapPoseCreation Property

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

Remarks If true, the device supports pose registration function.

If false, the device does not support pose registration function.

If this property is FALSE, you cannot use the **createPose** method to change the **PoseCreationMode** property.

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

Errors A UposException may be thrown when this method is invoked. For further information, see “**Errors**" on page Intro-20.

See Also **PoseCreationMode** Property, **createPose** Method.

MotionList Property

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

Remarks Comma-separated list of motion IDs defined on the device.

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

Errors A UposException may be thrown when this method is invoked. For further information, see “**Errors**" on page Intro-20.

PoseList Property

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

Remarks A comma-separated list of pause IDs defined on the device.

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

Errors A UposException may be thrown when this method is invoked. For further information, see “**Errors**" on page Intro-20.

PoseCreationMode Property

Syntax **PoseCreationMode:** ***boolean*** **{read-write, access after open, claim, enable}**

Remarks If true, pose registration function is enabled.

If false, pose registration function is invalid.

When this property is set to true, pause registration function is enabled. When false is set, the pause registration function is disabled.

This property is initialized to false when you first enable the device after calling the **open** method.

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified.

See Also **CapPose** Property, **CapPoseCreation** Property.

Methods (UML operations)

setPosition Method

Syntax **setPosition (positionList: *string*, time: *int32*, absolute: *boolean*):**
void {raises-exception, use after open, claim, enable}

Parameter	Description
positionList	Specify the position information in a comma-separated list.
time	Specify the time to control completion in seconds. If this value is too small, it will be changed to an appropriate value depending on the service.
absolute	If true, the specified position indicates the absolute value. If false, the specified position indicates relative value.

Each position information specified in the positionList consists of the following information and is shown in the following order separated by a colon (":").

Parameter	Description
jointID	Specify the joint ID. Specify one of the values listed in the **JointList** property. However, it must be an ID whose position range exists or not.
position	Specify the position to be set. Valid values range from -100 to 100. 100 represents the limit value in the positive direction of the target joint, and -100 represents the limit value in the negative direction. If Absolute is a relative value (false) and the value specified here exceeds the limit value, it will be changed to an appropriate value by the service

For example, to move Yow of Joint 01 up to the limit of the positive direction and move Pitch of Joint 02 to the middle, specify as follows.
"Joint01_Yaw:100,Joint01:Pitch:0"

Remarks The joint position is set with the contents specified in PositionList and control is started so that control is completed at the time specified by Time.

Joints that can be specified with this method are only those that have a position range.

Check the **JointList** property for the presence or absence of the position range.

This method is executed asynchronously. To terminate the operation prematurely, call the **stopControl** method.

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified.

See Also **JointList** Property, **stopControl** Method.

setSpeed Method

Syntax **setSpeed (speedList: *string*, time: *int32*):**
void {raises-exception, use after open, claim, enable}

Parameter	Description
speedList	Specify speed information in a comma-separated list.
time	Specify the time to control in seconds. If you specify FOREVER(-1), it will continue to operate until you call the **stopControl** method.

Each speed information specified in the SpeedList consists of the following information, and it is shown in the following order separated by a colon (":").

Parameter	Description
jointID	Specify the joint ID. Specify one of the values listed in the **JointList** property.
speed	Specify the speed to set. Valid values range from -100 to 100.100 represents the maximum speed in the positive direction of the target joint, and -100 represents the maximum speed in the negative direction.

For example, to move Wheel's X at the maximum speed in the positive direction and Y at the Wheel at half the speed in the negative direction, specify as follows.
"Wheel_X: 100, Wheel_Y: -50"

Remarks It sets the speed of the joint with the contents specified by speedList and performs control for the time specified by time.

This method is executed asynchronously. To terminate the operation prematurely, call the **stopControl** method.

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified.

See Also **JointList** Property, **stopControl** Method.

getPosition Method

Syntax **getPosition (jointID: *string*, position: *int32 by reference*):**
void {raises-exception, use after open, claim, enable}

Parameter	Description
jointID	Specify the joint ID. Specify one of the values listed in the **JointList** property. However, it must be an ID whose position range exists or not.
position	The position of the joint specified by JointID is stored.

Remarks It acquires the position specified by jointID and stores it in position.

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified.

See Also **JointList** Property.

startMotion Method

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

Parameter	Description
fileName	Specify the name of the motion file to start. Or one of the motion ID lists listed in the **MotionList** property.

Remarks Start motion defined by fileName or motion defined by the device.

Motion files need to be placed in the area managed by "hard total" service.

This method is executed asynchronously. To terminate motion control prematurely, call the **stopControl** method.

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified.
E_NOEXIST	File does not exist.

See Also **MotionList** Property.

createMotion Method

Syntax **createMotion (fileName: *string*, poseList: *string*):**
void {raises-exception, use after open, claim, enable}

Parameter	Description
fileName	Specify the motion file name to record motion.
poseList	Specify the comma-separated list of pause information to be registered.

Remarks Specify the registered pose and record it in the motion file.

The place where the motion file is recorded is the area managed by the "hard total" service.

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

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

Value	Meaning
E_ILLEGAL	fileName is too long or contains unusable characters.
E_EXISTS	fileName already exists.

startPose Method

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

Parameter	Description
fileName	Specify the name of the pause file to start. Or one of the pose ID lists listed in the **PoseList** property.

Remarks Begin pause defined by the pause file or device specified by fileName.

Pose files must be placed in the area managed by "hard total" service.

This method is executed asynchronously. To terminate pause control prematurely, call the **stopControl** method.

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified.
E_NOEXISTS	File does not exist.

See Also **PoseList** Property, **stopControl** Method.

createPose Method

Syntax **createPose (fileName: *string*, time: *int32*):**
void {raises-exception, use after open, claim, enable}

Parameter	Description
fileName	Specify the pose file name to record the pose.
time	Specify the time to reach the pose position.

Remarks Record the position of each joint in the pause file.

Before calling this method, you need to set the **PoseCreationMode** property to TRUE and enable pause registration mode.

The place where the pause file is recorded is the area managed by the "hard total" service.

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

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

Value	Meaning
E_ILLEGAL	FileName is too long or contains unusable characters. Or PoseCreationMode is FALSE.
E_EXISTS	FileName already exists.

See Also **PoseCreationMode** Property.

stopControl Method

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

Parameter	Description
outputID	Specify the value of the **OutputID** property you wish to terminate.

Remarks Stop the control specified for outputID.

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified.

See Also **setPosition** Method, **setSpeed** Method, **startPose** Method, **startMotion** Method.

CHAPTER 46

Device Monitor

This Chapter defines the Device Monitor device category.

Summary

Properties (UML attributes)

Common	Type	Mutability	*Version*	*May Use After*
AutoDisable:	*boolean*	{read-write}	1.16	open
CapCompareFirmwareVersion:	*boolean*	{read-only}	1.16	open
CapPowerReporting:	*int32*	{read-only}	1.16	open
CapStatisticsReporting:	*boolean*	{read-only}	1.16	open
CapUpdateFirmware:	*boolean*	{read-only}	1.16	open
CapUpdateStatistics:	*boolean*	{read-only}	1.16	open
CheckHealthText:	*string*	{read-only}	1.16	open
Claimed:	*boolean*	{read-only}	1.16	open
DataCount:	*int32*	{read-only}	1.16	open
DataEventEnabled:	*boolean*	{read-write}	1.16	open
DeviceEnabled:	*boolean*	{read-write}	1.16	open, claim
FreezeEvents:	*boolean*	{read-write}	1.16	open
OutputID:	*int32*	{read-only}	1.16	Not Supported
PowerNotify:	*int32*	{read-write}	1.16	open
PowerState:	*int32*	{read-only}	1.16	open
State:	*int32*	{read-only}	1.16	--
DeviceControlDescription:	*string*	{read-only}	1.16	--
DeviceControlVersion:	*int32*	{read-only}	1.16	--
DeviceServiceDescription:	*string*	{read-only}	1.16	open
DeviceServiceVersion:	*int32*	{read-only}	1.16	open
PhysicalDeviceDescription:	*string*	{read-only}	1.16	open
PhysicalDeviceName:	*string*	{read-only}	1.16	open

Properties (Continued)

Specific	*Type*	*Mutability*	*Version*	*May Use After*
DeviceList:	*string*	{read-only}	1.16	open
MonitoringDeviceList:	*string*	{read-only	1.16	open, claim & enable
DeviceData:	*string*	{read-only}	1.16	open, claim & enable

Methods (UML operations)

Common

Name	*Version*
open (logicalDeviceName: *string*`)`**:** **void {raises-exception}**	1.16
close ( ): **void {raises-exception, use after open}**	1.16
claim (timeout: *int32*`)`**:** **void {raises-exception, use after open}**	1.16
release ( ): **void {raises-exception, use after open, claim}**	1.16
checkHealth (level: *int32*`)`**:** **void {raises-exception, use after open, enable}**	1.16
clearInput ( ): **void { }**	Not supported
clearInputProperties ( ): **void { }**	Not supported
clearOutput ( ): **void { }**	Not supported
compareFirmwareVersion (firmwareFileName: *string*, **out result:** *int32*`)`**:** **void {raises-exception, use after open, enable}**	1.16
directIO (command: *int32*, **inout data:** *int32*, **inout obj:** *object*`)`**:** **void {raises-exception, use after open}**	1.16
resetStatistics (statisticsBuffer: *string*`)`**:** **void {raises-exception, use after open, enable}**	1.16
retrieveStatistics (inout statisticsBuffer: *string*`)`**:** **void {raises-exception, use after open, enable}**	1.16
updateFirmware (firmwareFileName: *string*`)`**:** **void {raises-exception, use after open, enable}**	1.16
updateStatistics (statisticsBuffer: *string*`)`**:** **void {raises-exception, use after open, enable}**	1.16

Specific

addMonitoringDevice (deviceID: *string*, **monitoringMode:** *int32,* **boundary:** *int32,* **subBoundary:** *int32,* **intervalTime:** *int32* **):** **void {raises-exception, use after open, claim, enable}**	1.16
deleteMonitoringDevice (deviceID: *string* **):** **void {raises-exception, use after open, claim, enable}**	1.16
clearMonitoringDevice ( ): **void {raises-exception, use after open, claim, enable}**	1.16
getDeviceValue (deviceID: *string*, **inout value:** *int32*) **void {raises-exception, use after open}**	1.16

Events (UML interfaces)

Name	*Type*	*Mutability*	*Version*
upos::events::DataEvent			1.16
Status:	*int32*	{read-only}	
upos::events::DirectIOEvent			1.16
EventNumber:	*int32*	{read-only}	
Data:	*int32*	{read-write}	
Obj:	*object*	{read-write}	
upos::events::ErrorEvent			1.16
ErrorCode:	*int32*	{read-only}	
ErrorCodeExtended:	*int32*	{read-only}	
ErrorLocus:	*int32*	{read-only}	
ErrorResponse:	*int32*	{read-write}	
upos::events::OutputCompleteEvent		***Not Supported***	
upos::events::StatusUpdateEvent			1.16
Status:	*int32*	{read-only}	

General Information

The Device Monitor programmatic name is "DeviceMonitor".

Capabilities

The Device Monitor Device has the following capability:

- Get values measured by various devices.
- Notify the application of changes in values measured by various devices.

Device Monitor Class Diagram

The following diagram shows the relationships between the Device Monitor classes.

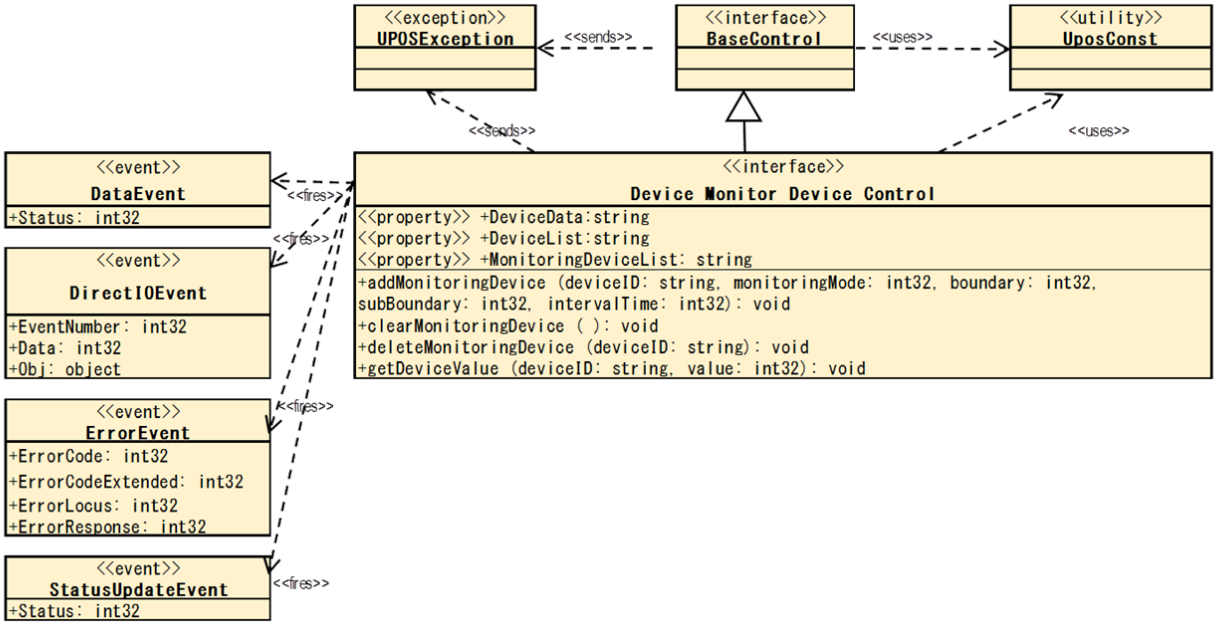


Fig. Chap. 46-1 Device Monitor Class Diagram

Model

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

- The Device Monitor supports monitoring of values measured by multiple devices connected to the device. A device that can be monitored and its type / value unit is listed in the **DeviceList** property.
- Device Monitor receives a change in the value measured by the device set as the monitoring target, and generates a **DataEvent** when it matches the specified condition.
- To add a device to be monitored, specify the monitoring mode with the **addMonitoringDevice** method and add it. For details on monitoring mode, see the description of **addMonitoringDevice** method.
- If the **AutoDisable** property is true, the device will automatically disable itself when a **DataEvent** is enqueued.
- An enqueued **DataEvent** can be delivered to the application when the **DataEventEnabled** property is true and other event delivery requirements are met. Just before delivering this event, data is copied into corresponding properties, and further data events are disabled by setting **DataEventEnabled** to false. This causes subsequent input data to be enqueued while the application processes the current input and associated properties. When the application has finished processing the current input and is ready for more data, it reenables events by setting **DataEventEnabled** to true.
- An **ErrorEvent** (or events) is enqueued if an error occurs while gathering or processing input, and is delivered to the application when **DataEventEnabled** is true and other event delivery requirements are met.
- The **DataCount** property can be read to obtain the total number of enqueued **DataEvents**.
- All enqueued input may be deleted by calling C**learInput**. See the C**learInput** method description for more details.
- All data properties that are populated as a result of firing a **DataEvent** or **ErrorEvent** can be set back to their default values by calling the **clearInputProperties** method.
- The notified data is stored in the **DeviceData** property.
- In the device control, the measured value of the device is managed with an integer value of int32 type, but some devices handle decimal values. In that case, you can calculate the actual value by dividing the measured value by the factor for each device that can be acquired with the **DeviceList** property.

Device Sharing

The Device Monitor is an exclusive-use device, as follows:

- The application must claim the device before enabling it.
- The application must claim and enable the device before the device begins reading input, or before calling methods that manipulate the device.

See the “Summary” table for precise usage prerequisites.

Properties (UML attributes)

DeviceList Property

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

Remarks Contains the comma-delimited list of device information that are supported by the device.

Each object information consists of the following information and is shown in the following order, separated by a colon (":").

Parameter	Description
DeviceID	Indicates a unique ID in the service that identifies the device.
Type	Indicates the device type. For example, if it is a touch sensor it is expressed as "TouchSensor" and so on. However, this value depends on the service.
Unit	Indicates the unit of value held by various devices. For example, it is expressed as "on / off" for a touch sensor, "rad / s" for a gyroscope. However, this value depends on the service.
Coefficient	Indicates the coefficient for calculating the actual measured value held by various devices. The **DeviceData** property and the measured value of the device that can be obtained with the **GetDeviceValue** method are expressed as integers, but by dividing this value by the coefficient it is the actual value. Example: Device value = 365, coefficient = 10, actual value = 36.5 For example, if one device supports one touch sensor and one gyroscope, it will be as follows. "Touch 01: Touch Sensor: ON/OFF: 1, GyroX: Gyroscope: rad/s: 100000, GyroY: Gyroscope: rad/s: 100000, GyroZ: Gyroscope: rad/s: 100000"

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

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **DeviceData** Property, **addMonitoringDevice** Method,

getDeviceValue Method

MonitoringDeviceList Property

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

Remarks Contains the comma-delimited list of monitoring information on registered devices that are supported by the device.

Each monitoring information consists of the following information and is shown in the following order, separated by a colon (":").

Parameter	Description
DeviceID	Registered devices ID.
Monitoring mode	Registered monitoring mode.
Boundary	Registered boundary value. This value is set to 0 when the monitoring mode does not require a boundary value.
Sub boundary	Registered sub boundary value. This value is set to 0 when the monitoring mode does not require a sub boundary value.
Interval	Registered interval. (millisecond)

For example, if you set monitoring targets as follows,

[Monitor target 1]

Device ID = Device 01, monitoring mode = SNS_MM_UPDATE,

boundary line = 0, sub boundary line = 0, interval time = 0

[Monitor target 2]

Device ID = Device 02, monitoring mode = SNS_MM_STRADDLED, boundary

line = 365, sub boundary line = 0, interval time = 500

The values shown are as follows.

"Device 01: 0: 0: 0: 0, Device 02: 1: 365: 0: 500"

This property is initialized by the **open** method. It is also updated by calling

addMonitoringDevice method, **deleteMonitoringDevice** method,

clearMonitoringDevice method.

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

See Also **addMonitoringDevice** Method, **deleteMonitoringDevice** Method, **clearMonitoringDevice** Method

DeviceData Property

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

Remarks Measurement information of the device that matches the condition registered by a**ddMonitoringDevice** method is set.

Each measurement information consists of the following information and is shown in the following order, separated by a colon (":").

Parameter	Description
DeviceID	The target device ID.
Measured value	Measurement value of the device. The measured value is represented by an integer type. To convert it to an actual value, divide the measured value by the coefficient acquired by the **DeviceList** property. For example,"Device01:365" Its value is set prior to a **DataEvent** being delivered to the application.

Errors A UposException may be thrown when this property is accessed. For further information, see **"Errors"** on page Intro-20.

Methods (UML operations)

addMonitoringDevice Method

Syntax **addMonitoringDevice (deviceID: *string*, monitoringMode:*int32*, boundary:*int32*, subBoundary:*int32*, intervalTime:*int32*) : void{raises-exception, use after open, claim, enable}**

Parameter	Description
deviceID	The deviceID of the monitored device. Valid values are one of the device ID lists listed in the **DeviceList** property.
monitoringMode	Specify the monitoring mode for monitoring.
boundary	Specify the boundary value to be monitored.
subBoundary	Specify the sub boundary value to be monitored. This value must be less than Boundary.
intervalTime	Specify the interval in milliseconds between the occurrence of the event and the start of the next monitoring.

The monitoring modes specified for MonitoringMode are as follows.

Value	Description
DM_MMODE_UPDATE	Every time the measured value of the target device is updated, an event is notified. When set to this mode, the values of the argument boundary and subBoundary are ignored.
DM_MMODE_STRADDLED	When the measured value of the target device crosses the value of the argument boundary, it notifies the event. In addition, when the measured value matches the value of boundary, it notifies the event even when it changes from the matched state. When set to this mode, the value of the argument SubBoundary is ignored.
DM_MMODE_HIGH	When the measured value of the target device becomes equal to or larger than the value of the argument Boundary, it notifies the event. Even if the measured value is updated and it was again equal to or greater than the value of boundary, we will notify the event each time. When set to this mode, the value of the argument subBoundary is ignored.

DM_MMODE_LOW

Notifies the event when the measured value of the target device becomes less than or equal to the value of the argument boundary. Even when the measured value is updated and it was again less than the value of boundary, we will notify the event each time. When set to this mode, the value of the argument subBoundary is ignored.

DM_MMODE_WITHIN

It notifies the event while the measured value of the target device is within the range specified by the argument boundary and subBoundary. Even if the measured value is updated and its value is within the range again, the event is notified each time.

DM_MMODE_OUTSIDE

It notifies the event while the measured value of the target device is outside the range specified by the argument boundary and subBoundary. Even if the measured value is updated and its value was out of range again, we will notify the event each time.

DM_MMODE_POLLING

It notifies the measured value of the target device at the interval specified by intervalTime. When set to this mode, the values of the argument boundary and subBoundary are ignored.

Remarks Add the device specified by deviceID to the monitoring target. The monitoring mode is specified for monitoringMode, but there are monitoring modes not supported by some devices. In that case, E_ILLEGAL is raised as the UPOS exception.

Devices added by this method will be added to the list of **MonitoringDeviceList** properties. If a device to be monitored is specified, it will be changed to a new condition. To exclude the added device from the monitoring target, call **deleteMonitoringDevice** method or **clearMonitoringDevice** method.

Errors A UposException may be thrown when this method is invoked. For further information, see **"Errors"** on page Intro-20. Some possible values of the exception's ErrorCode property are:

Value	Description
E_ILLEGAL	An invalid value was specified, or unsupported operation with the Device

See Also **DeviceList** Property, **MonitoringDeviceList** Property, **deleteMonitoringDevice** Method, **clearMonitoringDevice** Method

deleteMonitoringDevice Method

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

Parameter	Description
deviceID	Specify the device ID of the device to be excluded from monitoring targets.

Remarks Exclude the device specified by deviceID from monitoring targets.

Errors A UposException may be thrown when this method is invoked. For further information, see **"Errors"** on page Intro-20.
Some possible values of the exception's ErrorCode property are:

Value	Description
E_ILLEGAL	An invalid value was specified, or unsupported operation with the Device.

See Also **AddMonitoringDevice** Method

clearMonitoringDevice Method

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

Remarks Exclude all devices to be monitored.

Errors A UposException may be thrown when this method is invoked.

For further information, see "Errors" on page Intro-20.

See Also **addMonitoringDevice** method

getDeviceValue method

Syntax **getDeviceValue (deviceID: *string,* inout value: *int32*)**
: void{raises-exception, use after open}

Parameter	Description
deviceID	Specify the device ID of the device from which the measurement value is to be acquired. Specify one of the device ID lists listed in the **DeviceList** property.
value	Measured value obtained from the device.

Remarks Get the measured value of the device specified by deviceID. The retrieved value is stored in pValue.

Errors A UposException may be thrown when this method is invoked. For further information, see **"Errors"** on page Intro-20.
Some possible values of the exception's ErrorCode property are:

Valu	Description
E_ILLEGAL	An invalid value was specified, or unsupported operation with the Device.

See Also **DeviceList** Property

CHAPTER 47

Graphic Display

This Chapter defines the Graphic Display device category.

Summary

Properties (UML attributes)

Common	*Type*	*Mutability*	*Version*	*May Use After*
AutoDisable:	*boolean*	{read-write}	1.16	open
CapCompareFirmwareVersion:	*boolean*	{read-only}	1.16	open
CapPowerReporting:	*int32*	{read-only}	1.16	open
CapStatisticsReporting:	*boolean*	{read-only}	1.16	open
CapUpdateFirmware:	*boolean*	{read-only}	1.16	open
CapUpdateStatistics:	*boolean*	{read-only}	1.16	open
CheckHealthText:	*string*	{read-only}	1.16	open
Claimed:	*boolean*	{read-only}	1.16	open
DataCount:	*int32*	{read-only}	1.16	open
DataEventEnabled:	*boolean*	{read-write}	1.16	open
DeviceEnabled:	*boolean*	{read-write}	1.16	open, claim
FreezeEvents:	*boolean*	{read-write}	1.16	open
OutputID:	*int32*	{read-only}	1.16	open
PowerNotify:	*int32*	{read-write}	1.16	open
PowerState:	*int32*	{read-only}	1.16	open
State:	*int32*	{read-only}	1.16	--
DeviceControlDescription:	*string*	{read-only}	1.16	--
DeviceControlVersion:	*int32*	{read-only}	1.16	--
DeviceServiceDescription:	*string*	{read-only}	1.16	open
DeviceServiceVersion:	*int32*	{read-only}	1.16	open
PhysicalDeviceDescription:	*string*	{read-only}	1.16	open
PhysicalDeviceName:	*string*	{read-only}	1.16	open

Properties (Continued)

Specific	*Type*	*Mutability*	*Version*	*May Use After*
CapVolume:	*boolean*	{read-only}	1.16	open
CapBrightness:	*boolean*	{read-only}	1.16	open
Volume:	*int32*	{read-write}	1.16	open, claim & enable
Brightness:	*int32*	{read-write}	1.16	open, claim & enable
DisplayMode:	*int32*	{read-write}	1.16	open, claim & enable
CapImageTypeList:	*string*	{read-only}	1.16	open
CapVideoTypeList:	*string*	{read-only}	1.16	open
CapBack:	*boolean*	{read-only}	1.16	open
CapForward:	*boolean*	{read-only}	1.16	open
LoadStatus:	*int32*	{read-only}	1.16	open
URL:	*string*	{read-only}	1.16	open

Methods (UML operations)

Common

Name	*Version*
open (logicalDeviceName: *string*`)`**:** **void {raises-exception}**	1.16
close ( ): **void {raises-exception, use after open}**	1.16
claim (timeout: *int32*`)`**:** **void {raises-exception, use after open}**	1.16
release ( ): **void {raises-exception, use after open, claim}**	1.16
checkHealth (level: *int32*`)`**:** **void {raises-exception, use after open, enable}**	1.16
clearInput ( ): **void { }**	Not supported
clearInputProperties ( ): **void { }**	Not supported
clearOutput ( ): **void { }**	Not supported
compareFirmwareVersion (firmwareFileName: *string*, **out result:** *int32*`)`**:** **void {raises-exception, use after open, enable}**	1.16
directIO (command: *int32*, **inout data:** *int32*, **inout obj:** *object*`)`**:** **void {raises-exception, use after open}**	1.16

resetStatistics (statisticsBuffer: *string*): 1.16
void {raises-exception, use after open, enable}

retrieveStatistics (inout statisticsBuffer: *string*): 1.16
void {raises-exception, use after open, enable}

updateFirmware (firmwareFileName: *string*): 1.16
void {raises-exception, use after open, enable}

updateStatistics (statisticsBuffer: *string*): 1.16
void {raises-exception, use after open, enable}

Specific

Name	*Version*
loadImage (fileName: *string*): **void {raises-exception, use after open, claim, enable}**	1.16
playVideo (fileName: *string*, **loop:** *boolean*): **void { raises-exception, use after open, claim, enable}**	1.16
stopVideo ( ): void {raises-exception, use after open, claim, enable}	1.16
loadURL (uRL: ***string*** **): void {raises-exception, use after open, claim, enable}**	1.16
goBack ( ): void {raises-exception, use after open, claim, enable}	1.16
goForward ( ): void {raises-exception, use after open, claim, enable}	1.16
updatePage ( ): void {raises-exception, use after open, claim, enable}	1.16
cancelLoading ( ): void {raises-exception, use after open, claim, enable}	1.16

Events (UML interfaces)

Name	*Type*	*Mutability*	*Version*
upos::events::DataEvent			1.16
Status:	*int32*	{read-only}	
upos::events::DirectIOEvent			1.16
EventNumber:	*int32*	{read-only}	
Data:	*int32*	{read-write}	
Obj:	*object*	{read-write}	
upos::events::ErrorEvent			1.16
ErrorCode:	*int32*	{read-only}	
ErrorCodeExtended:	*int32*	{read-only}	
ErrorLocus:	*int32*	{read-only}	
ErrorResponse	*int32*	{read-write}	
upos::events::OutputCompleteEvent			1.16
OutputID:	*int32*	{read-only}	
upos::events::StatusUpdateEvent			1.16
Status:	*int32*	{read-only}	

General Information

The Graphic Display programmatic name is "GraphicDisplay".

Capabilities

The Graphic Display has the following capability:

Displays the specified image.

Play the specified movie.

Display the specified web page.

Notify the application of changes in the load status of the web page.

Graphics Display Class Diagram

The following diagram shows the relationships between the Graphic Display classes.

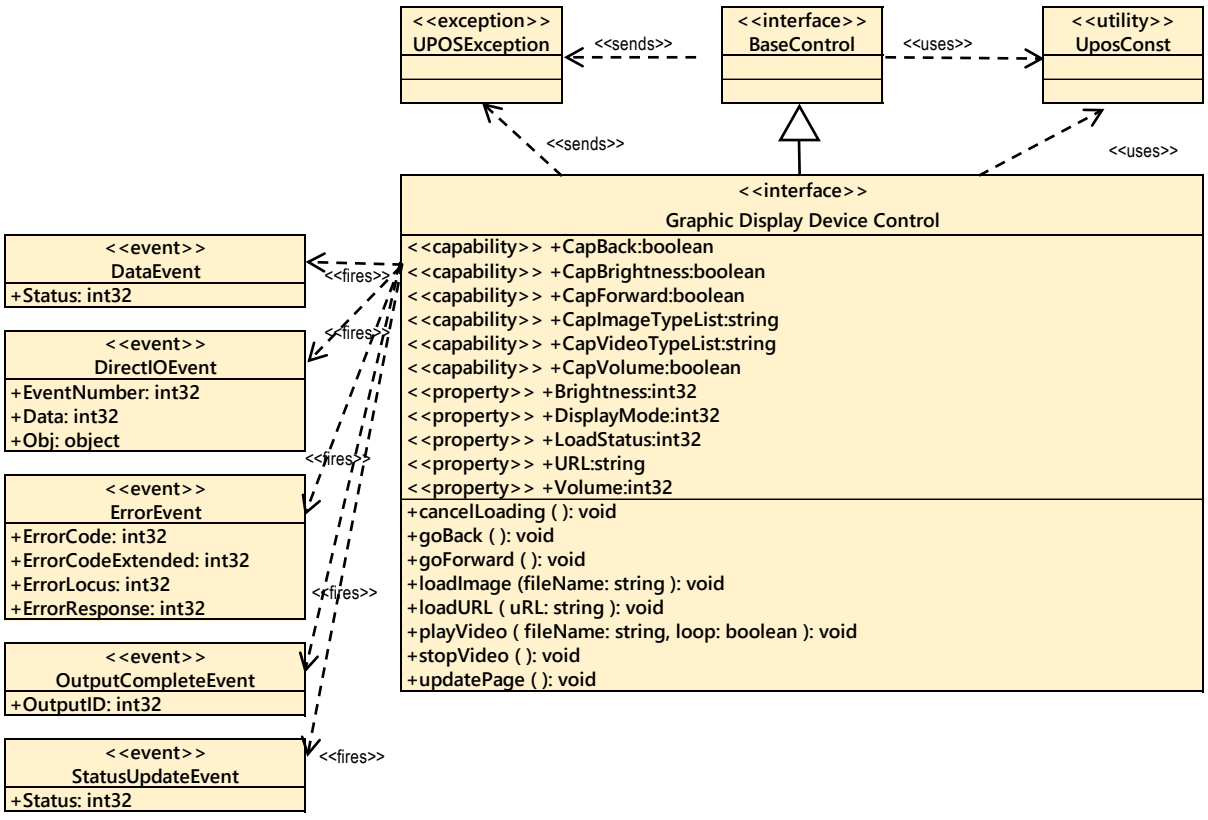


Fig. Chap. 47-1 Graphic Display Class Diagram

Model

The following display modes exist in the graphics control, and the model differs depending on the display mode:

- Image display mode
- Movie display mode.
- Web display mode.

The application can change the display mode by changing the value of the **DisplayMode** property.

Image Display Mode

The image display mode of the graphics control is as follows.

The application calls the **loadImage** method to display the image.

The **CapImageTypeList** property lists image files that the device can display.

Applications need to support “hard total” services as image files displaying with **loadImage** method must be placed in the area managed by the “hard total” service.

Movie Display Mode

The movie display mode of Graphic Display follows the general device behavior model for asynchronous output devices:

The application calls a **playVideo** method to start playing video. The Device validates the method parameters an error condition immediately if necessary. If the validation is successful, the Device does the following:

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

2. Sets the **OutputID** property to a unique integer identifier for this request.

3. Returns as soon as possible.

When the Device successfully completes a request, an **OutputCompleteEvent** is enqueued for delivery to the application.

A property of this event contains the output ID of the completed request.

The application should compare the returned **OutputCompleteEvent** property **OutputID** value with the **OutputID** value set by the asynchronous process method call used to send the data in order to track what data has been successfully sent to the device.

If an error occurs while processing a request, an **ErrorEvent** is enqueued which will be delivered to the application after the events already enqueued, including **OutputCompleteEvents**. No further asynchronous output will occur until the event has been delivered to the application. If the response is ER_CLEAR, then outstanding asynchronous output is cleared.

If the response is ER_RETRY, then output is retried; note that if several outputs were simultaneously in progress at the time that the error was detected, then the Service may need to retry all of these outputs.

Asynchronous output is always performed on a first-in first-out basis. If the device supports concurrent playback, the request will be executed simultaneously. To check if the device supports simultaneous playback, check the **CapMultiPlay** property.

If the request is terminated before completion, due to reasons such as the application calling the **clearOutput** method, then no **OutputCompleteEvent** is delivered. You can also delete the output individually by calling the **stopVideo** method. Also in this case **OutputCompleteEvent** will not be notified.

The **CapVideoTypeList** property lists video files that the device can play.

Applications need to support "hard total" services as video files played with the **playVideo** method must be placed in the area managed by the "hard total" service.

Web Display Mode

The web display mode of the Graphics Display follows the general "Device Input Model" for event-driven input:

When input is received from the Graphics Display, a **DataEvent** is enqueued.

If the **AutoDisable** property is true, then the device automatically disables itself when a **DataEvent** is enqueued.

An enqueued **DataEvent** can be delivered to the application when the **DataEventEnabled** property is true and other event delivery requirements are met. Just before delivering this event, data is copied into corresponding properties, and further data events are disabled by setting **DataEventEnabled** to false. This causes subsequent input data to be enqueued while the application processes the current input and associated properties. When the application has finished processing the current input and is ready for more data, it reenables events by setting **DataEventEnabled** to true.

An **ErrorEvent** (or events) is enqueued if an error occurs while gathering or processing input, and is delivered to the application when **DataEventEnabled** is true and other event delivery requirements are met.

The **DataCount** property may be read to obtain the total number of enqueued **DataEvents**.

All enqueued input may be deleted by calling **clearInput**. See the **clearInput** method description for more details.

All data properties that are populated as a result of firing a **DataEvent** or **ErrorEvent** can be set back to their default values by calling the **clearInputProperties** method.

The load state of the web page is stored in the **LoadStatus** property, and the URL is stored in the URL property.

Device Sharing

The web browser is an exclusive-use device, as follows:

- The application must claim the device before enabling it.
- The application must claim and enable the device before accessing some properties or calling methods that update the device.

See the "Summary" table for precise usage prerequisites.

Properties (UML attributes)

CapVolume Property

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

Remarks If true, the application can change the volume of video.

If false, the application cannot change the volume of video.

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

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

See Also **Volume** Property.

CapBrightness Property

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

Remarks If true, the application can change the screen brightness.

If false, the application cannot change the screen brightness.

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

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

See Also **Brightness** Property.

Volume Property

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

Remarks Holds the volume at playing video.

Legal values range from zero through 100.

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

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified.

See Also **CapVolume** Property**, playVideo** Method.

Brightness Property

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

Remarks Holds the brightness of screen.

Legal values range from zero through 100.

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

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified.

See Also **CapBrightness** Property.

DisplayMode Property

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

Remarks Holds the display mode.

Value	Meaning
GDISP_DMODE_HIDDEN	Hide the screen.
GDISP_DMODE_IMAGE_FIT	It is a mode to display images. The displayed image is enlarged / reduced to the size that maintains the aspect and just enters the screen.
GDISP_DMODE_IMAGE_FILL	It is a mode to display images. The displayed image is scaled to the size that maintains the aspect and covers the entire screen.
GDISP_DMODE_IMAGE_CENTER	It is a mode to display images. The displayed image is displayed in the center of the screen without changing the size.
GDISP_DMODE_VIDEO_NORMAL	It is a mode to display movies. The displayed movie will be displayed in the center of the screen without resizing it.
GDISP_DMODE_VIDEO_FULL	It is a mode to display movies. The displayed video will be displayed in full screen.
GDISP_DMODE_WEB	Display the web screen.

If application hide other modes and screens while displaying images, movies, or web, all displayed contents will be cleared. The movie will be stopped while the movie is playing.

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

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified.

See Also **CapCaptureColorSpaceList** Property, **VideoCaptureMode** Property, **readFrame** Method.

CapImageTypeList Property

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

Remarks Contains the comma-delimited list of image file type that are support by the device. For example, if the device only supports BMP and JPEG, then this property should be set to “BMP,JPEG”

*Notation contents may be different depending on the device.

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

Errors A UposException may be thrown when this method is invoked. For further information, see “**Errors**" on page Intro-20.

See Also **loadImage** Method.

CapVideoTypeList Property

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

Remarks Contains the comma-delimited list of video file type that are supported by the device. For example, if the device only supports AVI_IYUV and AVI_MJPG, then this property should be set to “AVI_IYUV,AVI_MJPG”.

*Notation contents may be different depending on the device.

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

Errors A UposException may be thrown when this method is invoked. For further information, see “**Errors**" on page Intro-20.

See Also **playVideo** Method.

CapBack Property

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

Remarks If true, the previous page exists in the browsing history. Application can return to the previous page with **goBack** method.

If false, there is no previous page in the browsing history.

This property is initialized to false by the open method. Also, as the web page loading state changes, it is set by the control.

Errors A UposException may be thrown when this method is invoked. For further information, see “**Errors**" on page Intro-20.

See Also **goBack** Method.

CapForward Property

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

Remarks If true, the next page exists in the browsing history. Application can go to the next page with the **goForward** method.

If false, there is no next page in the browsing history.

This property is initialized to false by the open method. Also, as the web page loading state changes, it is set by the control.

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

See Also **goForward** Method.

LoadStatus Property

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

Remarks Holds loading state of web page.

The parameters to be set are as follows.

Value	Meaning
GDISP_LSTATUS_START	Start loading the web page.
GDISP_LSTATUS_FINISH	It have finished loading the web page.
GDISP_LSTATUS_CANCEL	It have canceled loading the web page.

Its value is set prior to a **DataEvent** being delivered to the application.

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

URL Property

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

Remarks When the **LoadStatus** property is GDISP_LSTATUS_START, the URL of the Web page that starts loading is set.

When the **LoadStatus** property is GDISP_LSTATUS_FINISH, the URL of the loaded Web page is set.

When the **LoadStatus** property is GDISP_STATUS_CALCEL, the URL of the canceled Web page is set.

Its value is set prior to a **DataEvent** being delivered to the application.

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

See Also **loadStatus** Method.

Methods (UML operations)

loadImage Method

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

Parameter	Description
fileName	Specify the file name of the image to be loaded.

Remarks Load the specified image.

This method fails if the value of the **DisplayMode** Property is not set to GDISP_DMODE_IMAGE_FIT, GDISP_DMODE_IMAGE_FILL, or GDISP_DMODE_IMAGE_CENTER.

Image files must be located in the area managed by "Hard Total" service.

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified. Or an unsupported image file was specified.
E_NOEXIST	File does not exist.

See Also **DisplayMode** Property.

playVideo Method

Syntax **playVideo (fileName: *string*, loop: *boolean*):**
void {raises-exception, use after open, claim, enable}

Parameter	Description
fileName	Specify the file name of the video to be played.
loop	If true, loop playback is performed, and if false, loop playback is not performed.

Remarks Play the specified video.

If the value of the **DisplayMode** property is not set to GDISP_DMODE_VIDEO_NORMAL, GDISP_DMODE_VIDEO_FULL, this method will fail.

This method is executed asynchronously. To stop video playback in the middle, call the **stopVideo** method.

Video files must be located in the area managed by "Hard Total" service.

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified. Or an unsupported video file was specified.
E_NOEXIST	File does not exist.

See Also **DisplayMode** Property.

stopVideo Method

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

Remarks Stop the video being played.

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

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

Value	Meaning
E_ILLEGAL	The movie is not playing.

See Also **startVideo** Method.

loadURL Method

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

Parameter	Description
uRL	Specify the uRL of the web page to load.

Remarks Load the web page with the specified uRL.

This method is executed asynchronously. The load status is reported by **DataEvent** or **ErrorEvent**.

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

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

Value	Meaning
E_ILLEGAL	An invalid value was specified.

goBack Method

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

Remarks It returns to the previous page of browsing history.

This method is executed asynchronously. The load status is reported by **DataEvent** or **ErrorEvent**.

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

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

Value	Meaning
E_ILLEGAL	There is no previous page in the browsing history.

See Also **CapBack** Property.

goForward Method

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

Remarks Go to the next page of browsing history.

This method is executed asynchronously. The load status is reported by **DataEvent** or **ErrorEvent**.

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

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

Value	Meaning
E_ILLEGAL	There is no next page in the browsing history.

See Also **CapForward** Property.

updatePage Method

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

Remarks Reload the current web page.

This method is executed asynchronously. The load status is reported by **DataEvent** or **ErrorEvent**.

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

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

Value	Meaning
E_ILLEGAL	Web page loading.

cancelLoading Method

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

Remarks Cancel loading web page.

This method is executed asynchronously. The load status is reported by **DataEvent** or **ErrorEvent**.

Errors A UposException may be thrown when this method is invoked. For further information, see "**Errors**" on page Intro-20.

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

Value	Meaning
E_ILLEGAL	It is not loading.

Appendix K

Relationship to other OMG specification and activities

Robotics Domain Task Force

Activities in Robotics Domain Task Force

The OMG Robotics Domain Task Force (Robotics DTF) fosters the integration of robotics systems from modular components through the adoption of OMG standards. It recommends the adoption and extends OMG technologies that apply to the specific domain of robotics systems where no current baseline specifications exist, such as MDA for Robotics. The object technology is not solely limited to software but is extended to real objects. It also collaborates with other organizations for standardization, such as the one for home information appliances, and makes an open effort to increase interoperability in the field of robotics.

(https://www.omg.org/robotics/)

RoIS Specification

Robotic Interaction Service Framework [RoIS] defines several functional components for robotic interaction services.

Definitions related to locations of entities in robotic services will be described with Robotic Localization Service[RLS]. Definitions of status of components in services will be described in conjunction with Robotic Technology Component [RTC], Finite State Machine Component for RTC [FSM4RTC] and Unified Component Model for Distributed Real-Time and Embedded Systems [UCM].

RoIS specification seeks that specify a RoIS framework, on top of which various service robot applications are developed.

Scope of RoIS specification

They are summarized in the following items.

- Interface between service application and Human Robot Interaction (HRI) engine
- Interface to obtain information from HRI Engine according to the timing of the service application's needs (Query)
- Interface to receive information from HRI Engine triggered by real time events (Event notification / subscription / cancellation)
- Interface for instructions to control HRI Engine functions (Command)
- Definition of common messages for all HRI Engines

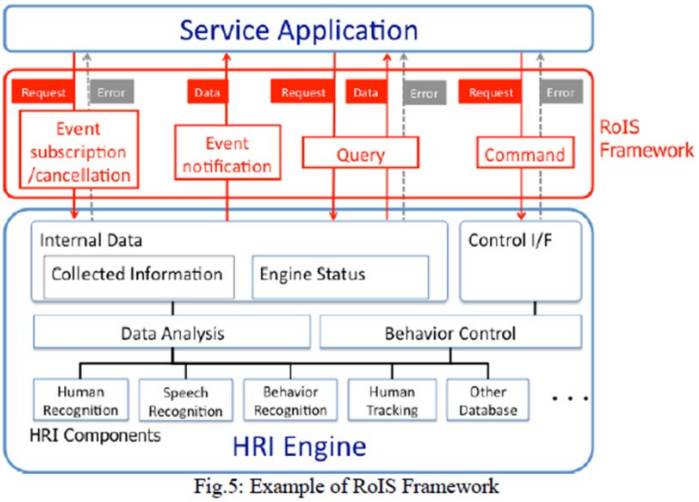


Fig.5: Example of RoIS Framework

Robot Servcie Ontology [RoSO] RFP

A new RFP of Robot Service Ontology[RoSO] currently being discussed in Robotics DTF are based on the concept of RoIS.

RoSO is aiming to define the specification (ontology) that clarifies the concept of a common vocabulary and / or a robot service in order to describe a service provided by a robot or exchange a description of a service provided by a service robot

Below is an example of HRI main component examples from this point of view.

Table K-1 – (From RoIS 1.2) Basic HRI Components

HRI Component Name	Description
system information	Provides the information of the system such as status of the system and position of the physical unit.
person detection	Detects number of people
person localization	Detects position of people
person identification	Identifies ID (name) of people
face detection	Detects number of human faces
face localization	Detects position of human faces
sound detection	Detects number of sound sources
sound localization	Detects position of sound sources
speech recognition	Recognizes person’s speech
gesture recognition	Recognizes person’s gesture
speech synthesis	Generates robot speech
reaction	Performs specified reaction
navigation	Moves to specified target location
follow	Follows a specified target object
move	Moves to specified distance or curve

Interoperability between UPOS RCSD and RoIs

RIeationsihp between UPOS RCSD and RoIS

OMG's Robotics standard provides a lower level control layer to manage Robot Device with finer granularity and higher accuracy to accommodate a wide range of industry applications.

On the other hand, the UPOS RCSD specification focuses on the functioning of robotic equipment within the retail store environment. In the UPOS RCSD specification robots are treated as peripheral equipment of the latest POS system. Therefore, the UPOS RCSD specification focuses on the definition of the interface between the POS and the robotic device.

RoIS is already existing as OMG standard and it defined a component frame service that was intended for robotic communication services with people.

Therefore, ROIS developed a general robot service framework, which is different from UPOS RCSD, but it is possible to describe the function of UPOS RCSD.

To confirm the compatibility and interoperability of the RCSD functions of RoIS and UPOS, both DTFs created and confirmed the function mapping table.

For this purpose, we use the general RoIS HRI component defined in the RoIS 1.2 specification.

UPOS RCSD Device and HRI Components Mapping Check Result

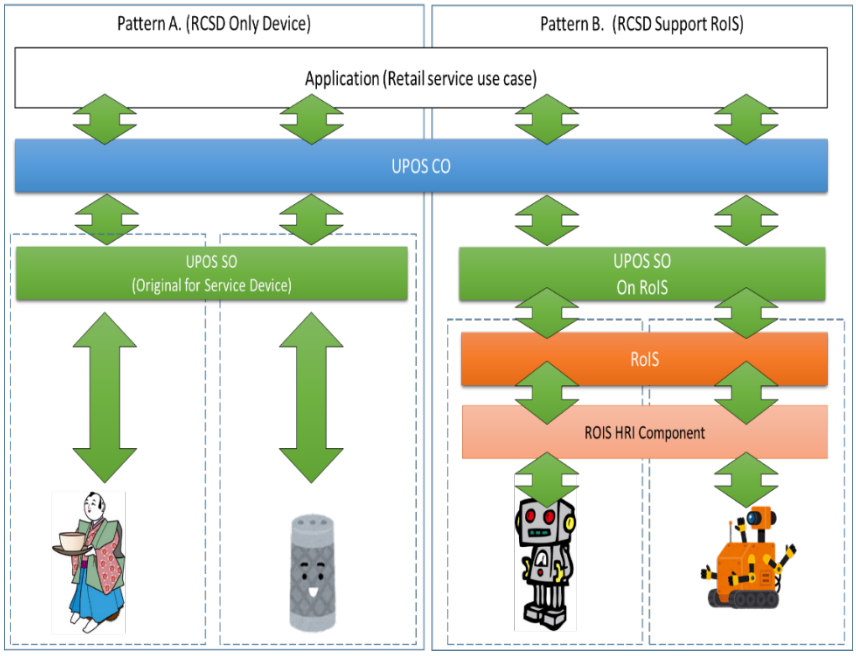
UPOS Device	RoIS HRI Component Name	Description
Capability(function) of each device	system information	Provides the information of the system such as status of the system and position of the physical unit.
Individual Recognition	person detection	Detects number of people
	person localization	Detects position of people
	person identification	Identifies ID (name) of people
	face detection	Detects number of human faces
	face localization	Detects position of human faces
	gesture recognition	Recognizes person's gesture
Sound & Voice Recognition	sound detection	Detects number of sound sources
	sound localization	Detects position of sound sources
	speech recognition	Recognizes person's speech
Speech Synthesis	speech synthesis	Generates robot speech
Gesture Control	reaction	Performs specified reaction
	navigation	Moves to specified target location
	follow	Follows a specified target object
	move	Moves to specified distance or curve
POS Power	Implementable as user defined Component	N/A
Lights		
Video Capture		
Sound Recorder		
Sound Player		
Device Monitor		
Graphic Display		

The two teams continue to collaborate between the part of their separate RFP's and standards that will be established.

For that purpose, it is very necessary to understand the common vocabulary of the robot service and the needs of the ontology.

If each team's specification satisfies the above mapping table, it is confirmed that the standard can be maintained independently.

In addition, the figure below shows a typical scenario where RCSD and RoIS work independently or in conjunction.



Document History

Version History

Ver	Date	Sections	Description of Change
1.0	2019-2-18		Initial Version – additions and updates to UPOS v1.15
			-

Glossary

Term	Definition
EVRW	Electronic Value Reader Writer
CAT	Credit Authorization Terminal