

Extensions for Financial Services (XFS)

XFS4IoT Specification Preview

This CEN Workshop Agreement has been drafted and approved by a Workshop of representatives of interested parties, the constitution of which is indicated in the foreword of this Workshop Agreement.

The formal process followed by the Workshop in the development of this Workshop Agreement has been endorsed by the National Members of CEN but neither the National Members of CEN nor the CEN-CENELEC Management Centre can be held accountable for the technical content of this CEN Workshop Agreement or possible conflicts with standards or legislation.

This CEN Workshop Agreement can in no way be held as being an official standard developed by CEN and its Members.

This CEN Workshop Agreement is publicly available as a reference document from the CEN Members National Standard Bodies.

CEN members are the national standards bodies of Austria, Belgium, Bulgaria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Republic of North Macedonia, Romania, Serbia, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey and United Kingdom.



EUROPEAN COMMITTEE FOR STANDARDIZATION
COMITÉ EUROPÉEN DE NORMALISATION
EUROPÄISCHES KOMITEE FÜR NORMUNG

Management Centre: rue de Stassart, 36 B-1050 Brussels

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Table of Contents

- 1 - API
 - 1.1 - Summary
 - 1.2 - General Information
 - 1.2.1 - Service Discovery
 - 1.2.2 - Messages
 - 1.2.3 - Command Sequence
 - **1.2.4 - End to End Security**
 - 1.3 - Commands
 - 1.3.1 - Common.GetService
 - 1.4 - Event Messages
 - 1.4.1 - Common.ServiceDetailEvent
- 2 - Common Interface
 - 2.1 - Summary
 - 2.2 - Commands
 - 2.2.1 - Common.Status
 - 2.2.2 - Common.Capabilities
 - 2.2.3 - Common.SetGuidanceLight
 - 2.2.4 - Common.PowerSaveControl
 - 2.2.5 - Common.SynchronizeCommand
 - 2.2.6 - Common.SetTransactionState
 - 2.2.7 - Common.GetTransactionState
 - 2.2.8 - Common.GetCommandNonce
 - 2.3 - Unsolicited Messages
 - 2.3.1 - Common.PowerSaveChangeEvent
 - 2.3.2 - Common.DevicePositionEvent
- 3 - Card Reader Interface
 - 3.1 - Summary
 - 3.2 - General Information
 - 3.2.1 - Intelligent Contactless Sequence Diagrams
 - 3.3 - Commands
 - 3.3.1 - CardReader.QueryIFMIdentifier
 - 3.3.2 - CardReader.EMVClassQueryApplications
 - 3.3.3 - CardReader.EjectCard
 - 3.3.4 - CardReader.RetainCard
 - 3.3.5 - CardReader.ResetCount
 - 3.3.6 - CardReader.SetKey
 - 3.3.7 - CardReader.ReadRawData
 - 3.3.8 - CardReader.WriteRawData

- 3.3.9 - CardReader.ChipIO
- 3.3.10 - CardReader.Reset
- 3.3.11 - CardReader.ChipPower
- 3.3.12 - CardReader.ParkCard
- 3.3.13 - CardReader.EMVClessConfigure
- 3.3.14 - CardReader.EMVClessPerformTransaction
- 3.3.15 - CardReader.EMVClessIssuerUpdate
- 3.4 - Event Messages
 - 3.4.1 - CardReader.MediaRetainedEvent
 - 3.4.2 - CardReader.InsertCardEvent
 - 3.4.3 - CardReader.MediaInsertedEvent
 - 3.4.4 - CardReader.InvalidMediaEvent
 - 3.4.5 - CardReader.TrackDetectedEvent
 - 3.4.6 - CardReader.InvalidTrackDataEvent
 - 3.4.7 - CardReader.MediaDetectedEvent
 - 3.4.8 - CardReader.EMVClessReadStatusEvent
- 3.5 - Unsolicited Messages
 - 3.5.1 - CardReader.MediaRemovedEvent
 - 3.5.2 - CardReader.CardActionEvent
 - 3.5.3 - CardReader.RetainBinThresholdEvent
- 4 - Cash Management Interface
 - 4.1 - Summary
 - 4.2 - General Information
 - 4.2.1 - Note Classification
 - 4.3 - Commands
 - 4.3.1 - CashManagement.GetCashUnitInfo
 - 4.3.2 - CashManagement.GetTellerInfo
 - 4.3.3 - CashManagement.GetItemInfo
 - 4.3.4 - CashManagement.GetClassificationList
 - 4.3.5 - CashManagement.SetTellerInfo
 - 4.3.6 - CashManagement.SetCashUnitInfo
 - 4.3.7 - CashManagement.OpenSafeDoor
 - 4.3.8 - CashManagement.StartExchange
 - 4.3.9 - CashManagement.EndExchange
 - 4.3.10 - CashManagement.CalibrateCashUnit
 - 4.3.11 - CashManagement.SetClassificationList
 - 4.4 - Event Messages
 - 4.4.1 - CashManagement.CashUnitErrorEvent
 - 4.4.2 - CashManagement.NoteErrorEvent
 - 4.4.3 - CashManagement.InfoAvailableEvent

- 4.4.4 - CashAcceptor.ShutterStatusChangedEvent
- 4.4.5 - Dispenser.ItemsTakenEvent
- 4.5 - Unsolicited Messages
 - 4.5.1 - CashManagement.SafeDoorOpenEvent
 - 4.5.2 - CashManagement.SafeDoorClosedEvent
 - 4.5.3 - CashManagement.CashUnitInfoChangedEvent
 - 4.5.4 - CashManagement.TellerInfoChangedEvent
 - 4.5.5 - CashManagement.CashUnitThresholdEvent
- 5 - Dispenser Interface
 - 5.1 - Summary
 - 5.2 - Commands
 - 5.2.1 - Dispenser.GetMixTypes
 - 5.2.2 - Dispenser.GetMixTable
 - 5.2.3 - Dispenser.GetPresentStatus
 - 5.2.4 - Dispenser.Denominate
 - 5.2.5 - Dispenser.Dispense
 - 5.2.6 - Dispenser.Present
 - 5.2.7 - Dispenser.Reject
 - 5.2.8 - Dispenser.Retract
 - 5.2.9 - Dispenser.OpenShutter
 - 5.2.10 - Dispenser.CloseShutter
 - 5.2.11 - Dispenser.SetMixTable
 - 5.2.12 - Dispenser.Reset
 - 5.2.13 - Dispenser.TestCashUnits
 - 5.2.14 - Dispenser.Count
 - 5.2.15 - Dispenser.PrepareDispense
 - 5.3 - Event Messages
 - 5.3.1 - CashManagement.CashUnitErrorEvent
 - 5.3.2 - CashManagement.CashUnitThresholdEvent
 - 5.3.3 - Dispenser.DelayedDispenseEvent
 - 5.3.4 - Dispenser.StartDispenseEvent
 - 5.3.5 - Dispenser.PartialDispenseEvent
 - 5.3.6 - Dispenser.SubDispenseOkEvent
 - 5.3.7 - Dispenser.IncompleteDispenseEvent
 - 5.3.8 - CashManagement.NoteErrorEvent
 - 5.3.9 - CashManagement.InfoAvailableEvent
 - 5.3.10 - Dispenser.IncompleteRetractEvent
 - 5.3.11 - CashManagement.CashUnitInfoChangedEvent
 - 5.4 - Unsolicited Messages
 - 5.4.1 - Dispenser.ItemsTakenEvent

- 5.4.2 - Dispenser.ItemsPresentedEvent
- 5.4.3 - Dispenser.MediaDetectedEvent
- 5.4.4 - Dispenser.ShutterStatusChangedEvent
- **6 - Cash Acceptor Interface**
 - 6.1 - Summary
 - 6.2 - Commands
 - 6.2.1 - CashAcceptor.GetBanknoteTypes
 - 6.2.2 - CashAcceptor.GetCashInStatus
 - 6.2.3 - CashAcceptor.GetPositionCapabilities
 - 6.2.4 - CashAcceptor.GetReplenishTarget
 - 6.2.5 - CashAcceptor.GetDeviceClockStatus
 - 6.2.6 - CashAcceptor.GetCashUnitCapabilities
 - 6.2.7 - CashAcceptor.GetDepleteSource
 - 6.2.8 - CashAcceptor.GetCashUnitCountStatus
 - 6.2.9 - CashAcceptor.GetPresentStatus
 - 6.2.10 - CashAcceptor.CashInStart
 - 6.2.11 - CashAcceptor.CashIn
 - 6.2.12 - CashAcceptor.CashInEnd
 - 6.2.13 - CashAcceptor.CashInRollback
 - 6.2.14 - CashAcceptor.Retract
 - 6.2.15 - CashAcceptor.OpenShutter
 - 6.2.16 - CashAcceptor.CloseShutter
 - 6.2.17 - CashAcceptor.Reset
 - 6.2.18 - CashAcceptor.ConfigureNoteTypes
 - 6.2.19 - CashAcceptor.CreateP6Signature
 - 6.2.20 - CashAcceptor.ConfigureNoteReader
 - 6.2.21 - CashAcceptor.CompareP6Signature
 - 6.2.22 - CashAcceptor.Replenish
 - 6.2.23 - CashAcceptor.SetCashInLimit
 - 6.2.24 - CashAcceptor.CashUnitCount
 - 6.2.25 - CashAcceptor.DeviceLockControl
 - 6.2.26 - CashAcceptor.SetMode
 - 6.2.27 - CashAcceptor.PresentMedia
 - 6.2.28 - CashAcceptor.Deplete
 - 6.2.29 - CashAcceptor.PreparePresent
 - 6.3 - Event Messages
 - 6.3.1 - CashManagement.CashUnitErrorEvent
 - 6.3.2 - CashAcceptor.InputRefuseEvent
 - 6.3.3 - CashManagement.NoteErrorEvent
 - 6.3.4 - CashAcceptor.SubCashInEvent

- 6.3.5 - CashManagement.InfoAvailableEvent
- 6.3.6 - CashAcceptor.InsertItemsEvent
- 6.3.7 - CashManagement.CashUnitThresholdEvent
- 6.3.8 - CashManagement.CashUnitInfoChangedEvent
- 6.3.9 - CashAcceptor.IncompleteReplenishEvent
- 6.3.10 - CashAcceptor.IncompleteDepleteEvent
- 6.4 - Unsolicited Messages
 - 6.4.1 - CashAcceptor.ItemsTakenEvent
 - 6.4.2 - CashAcceptor.ItemsPresentedEvent
 - 6.4.3 - CashAcceptor.ItemsInsertedEvent
 - 6.4.4 - CashAcceptor.MediaDetectedEvent
 - 6.4.5 - CashAcceptor.ShutterStatusChangedEvent
 - 6.4.6 - CashAcceptor.CountAccuracyChangedEvent
- 7 - Key Management Interface
 - 7.1 - Summary
 - 7.2 - General Information
 - 7.2.1 - RKL Terminology
 - 7.2.2 - Remote Key Loading Using Signatures
 - 7.2.3 - Initialization Phase – Signature Issuer and ATM PIN
 - 7.2.4 - Initialization Phase – Signature Issuer and Host
 - 7.2.5 - Key Exchange – Host and ATM PIN
 - 7.2.6 - Key Exchange (with random number) – Host and ATM PIN
 - 7.2.7 - Enhanced RKL, Key Exchange (with random number) – Host and ATM PIN
 - 7.2.8 - Remote Key Loading Using Certificates
 - 7.2.9 - Certificate Exchange and Authentication
 - 7.2.10 - Remote Key Exchange
 - 7.2.11 - Replace Certificate
 - 7.2.12 - Primary and Secondary Certificate
 - 7.2.13 - TR34 BIND To Host
 - 7.2.14 - TR34 Key Transport
 - 7.2.15 - TR34 REBIND To New Host
 - 7.2.16 - TR34 Force REBIND To New Host
 - 7.2.17 - TR34 UNBIND From Host
 - 7.2.18 - TR34 Force UNBIND From Host
 - 7.2.19 - EMV Support
 - 7.2.20 - ImportKey command Input-Output Parameters
 - 7.2.21 - TR-31 Key Use
 - 7.2.22 - RestrictedKeyEndKey Command Usage
 - 7.3 - Commands
 - 7.3.1 - KeyManagement.GetKeyDetail

- 7.3.2 - KeyManagement.Initialization
- 7.3.3 - KeyManagement.DeriveKey
- 7.3.4 - KeyManagement.Reset
- 7.3.5 - KeyManagement.ImportKey
- 7.3.6 - KeyManagement.DeleteKey
- 7.3.7 - KeyManagement.ExportRSAIssuerSignedItem
- 7.3.8 - KeyManagement.GenerateRSAKeyPair
- 7.3.9 - KeyManagement.ExportRSAEPPSSignedItem
- 7.3.10 - KeyManagement.GetCertificate
- 7.3.11 - KeyManagement.ReplaceCertificate
- 7.3.12 - KeyManagement.StartKeyExchange
- 7.3.13 - KeyManagement.GenerateKCV
- 7.3.14 - KeyManagement.LoadCertificate
- 7.3.15 - KeyManagement.StartAuthenticate
- 7.4 - Unsolicited Messages
 - 7.4.1 - KeyManagement.InitializedEvent
 - 7.4.2 - KeyManagement.IllegalKeyAccessEvent
 - 7.4.3 - KeyManagement.CertificateChangeEvent
- 8 - Crypto Interface
 - 8.1 - Summary
 - 8.2 - General Information
 - 8.2.1 - DUKPT
 - 8.3 - Commands
 - 8.3.1 - Crypto.GenerateRandom
 - 8.3.2 - Crypto.CryptoData
 - 8.3.3 - Crypto.GenerateAuthentication
 - 8.3.4 - Crypto.VerifyAuthentication
 - 8.3.5 - Crypto.Digest
 - 8.4 - Event Messages
 - 8.4.1 - Pinpad.DUKPTKSNEvent
 - 8.5 - Unsolicited Messages
 - 8.5.1 - Crypto.IllegalKeyAccessEvent
- 9 - Keyboard Interface
 - 9.1 - Summary
 - 9.2 - General Information
 - 9.2.1 - Encrypting Touch Screen (ETS)
 - 9.2.2 - Secure Key Entry
 - 9.2.3 - Command Usage
 - 9.3 - Commands
 - 9.3.1 - Keyboard.GetFuncKeyDetail

- 9.3.2 - Keyboard.GetLayout
- 9.3.3 - Keyboard.PinEntry
- 9.3.4 - Keyboard.DataEntry
- 9.3.5 - Keyboard.Reset
- 9.3.6 - Keyboard.SecureKeyEntry
- 9.3.7 - Keyboard.KeypressBeep
- 9.3.8 - Keyboard.DefineLayout
- 9.4 - Event Messages
 - 9.4.1 - Keyboard.KeyEvent
 - 9.4.2 - Keyboard.EnterDataEvent
 - 9.4.3 - Keyboard.LayoutEvent
- 10 - Pinpad Interface
 - 10.1 - Summary
 - 10.2 - General Information
 - 10.2.1 - DUKPT
 - 10.3 - Commands
 - 10.3.1 - Pinpad.GetQueryPCIPTSDeviceId
 - 10.3.2 - Pinpad.LocalDES
 - 10.3.3 - Pinpad.LocalVisa
 - 10.3.4 - Pinpad.PresentIDC
 - 10.3.5 - Pinpad.Reset
 - 10.3.6 - Pinpad.MaintainPin
 - 10.3.7 - Pinpad.SetPinBlockData
 - 10.3.8 - Pinpad.GetPinBlock
 - 10.4 - Event Messages
 - 10.4.1 - Pinpad.DUKPTKSNEvent
 - 10.5 - Unsolicited Messages
 - 10.5.1 - Pinpad.IllegalKeyAccessEvent
- 11 - Printer Interface
 - 11.1 - Summary
 - 11.2 - General Information
 - 11.2.1 - Banking Printer Types
 - 11.2.2 - Forms Model
 - 11.2.3 - Command Overview
 - 11.2.4 - Form, Sub-Form, Field, Frame, Table and Media Definitions
 - 11.2.5 - Command and Event Flows during Single and Multi-Page / Wad Printing
 - 11.3 - Commands
 - 11.3.1 - Printer.GetFormList
 - 11.3.2 - Printer.GetMediaList
 - 11.3.3 - Printer.GetQueryForm

- 11.3.4 - Printer.GetQueryMedia
- 11.3.5 - Printer.GetQueryField
- 11.3.6 - Printer.GetCodelineMapping
- 11.3.7 - Printer.ControlMedia
- 11.3.8 - Printer.PrintForm
- 11.3.9 - Printer.ReadForm
- 11.3.10 - Printer.RawData
- 11.3.11 - Printer.MediaExtents
- 11.3.12 - Printer.ResetCount
- 11.3.13 - Printer.ReadImage
- 11.3.14 - Printer.Reset
- 11.3.15 - Printer.RetractMedia
- 11.3.16 - Printer.DispensePaper
- 11.3.17 - Printer.PrintRawFile
- 11.3.18 - Printer.LoadDefinition
- 11.3.19 - Printer.SupplyReplenish
- 11.3.20 - Printer.ControlPassbook
- 11.3.21 - Printer.SetBlackMarkMode
- 11.4 - Event Messages
 - 11.4.1 - Printer.MediaPresentedEvent
 - 11.4.2 - Printer.NoMediaEvent
 - 11.4.3 - Printer.MediaInsertedEvent
 - 11.4.4 - Printer.FieldErrorEvent
 - 11.4.5 - Printer.FieldWarningEvent
 - 11.4.6 - Printer.MediaRejectedEvent
- 11.5 - Unsolicited Messages
 - 11.5.1 - Printer.MediaTakenEvent
 - 11.5.2 - Printer.MediaInsertedUnsolicitedEvent
 - 11.5.3 - Printer.MediaPresentedUnsolicitedEvent
 - 11.5.4 - Printer.MediaDetectedEvent
 - 11.5.5 - Printer.RetractBinStatusEvent
 - 11.5.6 - Printer.DefinitionLoadedEvent
 - 11.5.7 - Printer.MediaAutoRetractedEvent
 - 11.5.8 - Printer.RetractBinThresholdEvent
 - 11.5.9 - Printer.PaperThresholdEvent
 - 11.5.10 - Printer.TonerThresholdEvent
 - 11.5.11 - Printer.LampThresholdEvent
 - 11.5.12 - Printer.InkThresholdEvent
- 12 - Text Terminal Interface
 - 12.1 - Summary

- 12.2 - General Information
 - 12.2.1 - Form and Field Definitions
- 12.3 - Commands
 - 12.3.1 - TextTerminal.GetFormList
 - 12.3.2 - TextTerminal.GetQueryForm
 - 12.3.3 - TextTerminal.GetQueryField
 - 12.3.4 - TextTerminal.GetKeyDetail
 - 12.3.5 - TextTerminal.Beep
 - 12.3.6 - TextTerminal.ClearScreen
 - 12.3.7 - TextTerminal.DispLight
 - 12.3.8 - TextTerminal.SetLed
 - 12.3.9 - TextTerminal.SetResolution
 - 12.3.10 - TextTerminal.WriteForm
 - 12.3.11 - TextTerminal.ReadForm
 - 12.3.12 - TextTerminal.Write
 - 12.3.13 - TextTerminal.Read
 - 12.3.14 - TextTerminal.Reset
 - 12.3.15 - TextTerminal.DefineKeys
- 12.4 - Unsolicited Messages
 - 12.4.1 - TextTerminal.FieldErrorEvent
 - 12.4.2 - TextTerminal.FieldWarningEvent
 - 12.4.3 - TextTerminal.KeyEvent
- 13 - Sensors and Indicators Interface
 - 13.1 - Summary
 - 13.2 - Commands
 - 13.2.1 - SensorsAndIndicators.SetGuidanceLight
 - 13.2.2 - SensorsAndIndicators.GetAutoStartupTime
 - 13.2.3 - SensorsAndIndicators.ClearAutoStartupTime
 - 13.2.4 - SensorsAndIndicators.Register
 - 13.2.5 - SensorsAndIndicators.SetPorts
 - 13.2.6 - SensorsAndIndicators.SetDoor
 - 13.2.7 - SensorsAndIndicators.SetIndicator
 - 13.2.8 - SensorsAndIndicators.SetAutostartupTime
 - 13.3 - Unsolicited Messages
 - 13.3.1 - SensorsAndIndicators.PortErrorEvent
- 14 - Barcode Reader Interface
 - 14.1 - Summary
 - 14.2 - Commands
 - 14.2.1 - BarcodeReader.Read
 - 14.2.2 - BarcodeReader.Reset

- [15 - Card Embosser Interface](#)
 - [15.1 - Summary](#)
 - [15.2 - General Information](#)
 - [15.2.1 - Embossing Form, Field and Media Definitions](#)
 - [15.3 - Commands](#)
 - [15.3.1 - CardEmbosser.GetFormList](#)
 - [15.3.2 - CardEmbosser.GetQueryForm](#)
 - [15.3.3 - CardEmbosser.GetMediaList](#)
 - [15.3.4 - CardEmbosser.GetQueryMedia](#)
 - [15.3.5 - CardEmbosser.GetQueryField](#)
 - [15.3.6 - CardEmbosser.EmbossCard](#)
 - [15.3.7 - CardEmbosser.Reset](#)
 - [15.3.8 - CardEmbosser.SupplyReplenish](#)
 - [15.4 - Event Messages](#)
 - [15.4.1 - CardEmbosser.EmbossFailureEvent](#)
 - [15.4.2 - CardEmbosser.FieldErrorEvent](#)
 - [15.4.3 - CardEmbosser.FieldWarningEvent](#)
 - [15.5 - Unsolicited Messages](#)
 - [15.5.1 - CardEmbosser.InputBinThresholdEvent](#)
 - [15.5.2 - CardEmbosser.OutputBinThresholdEvent](#)
 - [15.5.3 - CardEmbosser.RetainBinThresholdEvent](#)
 - [15.5.4 - CardEmbosser.MediaRemovedEvent](#)
 - [15.5.5 - CardEmbosser.MediaDetectedEvent](#)
 - [15.5.6 - CardEmbosser.TonerThresholdEvent](#)
- [16 - Biometric Interface](#)
 - [16.1 - Summary](#)
 - [16.2 - General Information](#)
 - [16.2.1 - Enrollment](#)
 - [16.2.2 - Biometric Matching](#)
 - [16.2.3 - Biometric Device Types](#)
 - [16.2.4 - Biometric Data Security](#)
 - [16.2.5 - Biometric Device Command Flows](#)
 - [16.3 - Commands](#)
 - [16.3.1 - Biometric.GetStorageInfo](#)
 - [16.3.2 - Biometric.Read](#)
 - [16.3.3 - Biometric.Import](#)
 - [16.3.4 - Biometric.Match](#)
 - [16.3.5 - Biometric.SetMatch](#)
 - [16.3.6 - Biometric.Clear](#)
 - [16.3.7 - Biometric.Reset](#)

- [16.3.8 - Biometric.SetDataPersistance](#)
- [16.4 - Unsolicited Messages](#)
 - [16.4.1 - Biometric.PresentSubjectEvent](#)
 - [16.4.2 - Biometric.SubjectDetectedEvent](#)
 - [16.4.3 - Biometric.RemoveSubjectEvent](#)
 - [16.4.4 - Biometric.SubjectRemovedEvent](#)
 - [16.4.5 - Biometric.DataClearedEvent](#)
 - [16.4.6 - Biometric.OrientationEvent](#)

1 - API

This chapter defines the API functionality and messages.

1.1 - Summary

This defines the XFS4IoT API including but not limited to:

- * Service Discovery
- * Message Definition
- * End to End Security

1.2 - General Information

1.2.1 - Service Discovery

This section covers the process for discovering services before they can be used.

1.2.1.1 - *Introduction*

Multiple services can be supplied by multiple vendors. This standard doesn't require coordination between these different organization, or between the service publishers and the service client. It is possible to operate a system with components from multiple hardware vendors, and with third party clients, without the prior knowledge of any party.

This specification covers an environment using WebSockets to communicate between services and clients, either on a single machine or across a network.

This section covers both the process for publishing a service such that it can be discovered, and the discovery process used by the service client.

There is also a clear definition of responsibility for each component, including when there are dependencies between components. There are no shared components required to coordinate the system.

The underlying network can use any protocol that supports WebSockets such as IPv4 or IPv6. Nothing in this document requires any particular underlying protocol.

This document uses CAN, WILL, MUST, SHOULD etc. in the normal ways to distinguish between requirements, recommendations, permissions and possibilities.

Requirements of this spec will be formatted as follows:

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

This is an example requirement

All other text is commentary used to illustrate the reasoning of the requirements. Where there is any conflict the requirement text always takes priority.

1.2.1.2 - Overview

In this standard there are two types of "end-point"; publisher and service. Each end-point, of either type, is published by a single software/hardware vendor. A publisher end-point is used for service discovery, to discover service end-points. A single service end-point can expose multiple "services", where each service typically represents a single piece of hardware. A single machine (or a single IP address) may expose multiple publisher and service end-points from different vendors. A "client" application may consume multiple services from multiple service end-points on the same machine, or across multiple machines.

On startup of the machine, any software services attempt to claim access to individual network ports using the underlying operating system mechanism. Ports are claimed sequentially from a known sequence. Each port becomes an end-point that can publish multiple services from a single vendor.

A client application will attempt to connect to each port on a machine in the known sequence to get a list of all active publisher end-points. For each publisher end-point it then exchanges JSON messages across WebSockets with URIs using a known format to recover a list of services published by that end-point. Once it has a full list of services it can use WebSocket connections to communicate with each service to perform whichever actions are required.

1.2.1.3 - Machine Identification

Machines publishing services are identified by URIs. Machines exposing end-points can be identified by an IP address or by a DNS name.

Either the IP address or DNS name for a machine must be known by the client for the client to connect. This would probably be a configuration setting for the client and would need to be known by the organization setting up the client, but this configuration is outside the scope of this document.

1.2.1.4 - Network Protocol

TLS security will be used to secure network connections. The only exception will be when the network connection between the client and service can be physically secured because they are both inside the same cabinet. In that case it will be possible to use clear communication without TLS encryption.

The publisher will publish all WebSocket services protected by TLS encryption. This will be identified by the wss:// protocol specifier.

The publisher may publish WebSocket services without TLS encryption, as a clear WebSocket connection, but only if the physical connection between the service and the client is physically protected. It is up to the hardware manufacturer to ensure this physical protection is sufficient. This unsecured connection will be identified by the ws:// protocol specifier.

Where TLS is used, the service will be protected by a mutually trusted server side certificate as part of the TLS protocol. This complete certificate chain must be mutually trusted by the client and service.

Establishing and managing the certificates between the service and the client is outside of the scope of this spec but trust must be in place. This might be achieved using a public third party certificate authority that issues TLS certificates. Alternatively it might be achieved using a bank's own internal CA. It shouldn't depend on a private CA or certificates issued by a vendor, which might limit access to the service.

A WSS connections with invalid certificates will be invalid and will be rejected by both the client and the service.

1.2.1.5 - URI Format

Communications are defined by URI's

Communication with service publishers and services will be through distinct URIs which will use the following format

wss://machinename:portnumber/xfs4iot/v1.0/servicename

This consists of the following parts:

wss:// or ws://

The protocol id for secure WebSockets. This should be wss:// for secure connections. An insecure ws:// connection can be used when the connection is physically secured, inside an ATM enclosure.

machinename

The identification of the machine publishing end-points. This can be an IP address or DNS name.

portnumber

The port number discovered through the initial service discovery process

XFS4IoT

A literal string. The inclusion of this part identifies standard XFS4IoT services published on this URI. It allows the possibility of a single vendor publishing standard and non-standard

proprietary services on the same port. Any standard service URI will start with this string. Any non-standard service's URI must not start with this string.

v1.0

The version of the XFS4IoT specification being used by this service. This will be updated in future versions of the specification and allows support for multiple versions of the specification on the same machine and end-point.

Note that most future changes to the XFS4IoT specification will be done in a non-breaking, backwards and forwards compatible way. For example, optional fields will be added to JSON messages when required. This means that changes to the version field of the URI will be very rare. It will only be changed if there is a breaking, incompatible change or a fundamental change to the API. Because of this there won't be any need for complex version negotiation between the client and the service. The client will simply attempt to open the version of the API that it supports.

ServiceName

This will be included in the URI to allow different services to be identified on the same port. Services will normally match individual devices. The exact service name is discovered during service discovery and is vendor dependent. The format of the service name shouldn't be assumed. The only URI that doesn't include a service name is the service discovery URI.

For example, a service discovery URI might be;

- wss://terminal321.atmnetwork.corporatenet:443/xfs4iot/v1.0
- wss://192.168.21.43:5848/xfs4iot/v1.0

Service URI might be;

- wss://terminal321.atmnetwork.corporatenet:443/xfs4iot/v1.0/maincashdispenser
- wss://192.168.21.43:5848/xfs4iot/v1.0/cardreader1

The URI will be case sensitive. The URI will be lower case.

1.2.1.6 - Service Publishing

Service publishers will negotiate access to resources and publish services using the following process.

Port Sequence

Services will be published on a sequence of IP ports consisting of two ranges consisting of the port 80 and 443 followed by the ports 5846 to 5856 (inclusive). Hence the full sequence of ports will be 12 ports as,

80 or 443, 5846, 5847, 5848, ... 5855, 5856

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Port 80 will only be used with HTTP/WS. Port 443 will only be used with HTTPS/WSS. All other ports may be used with either or both HTTP/WS and HTTPS/WSS.

Port 80 and 443 are the standard ports for HTTP and HTTPS and have the advantage that they are likely to be open on firewalls. The correct port will be used to match the protocol - 80 for HTTP/WS and 443 for HTTPS/WSS. Other ports are flexible and can be used for either protocol by the Service Publisher.

The port range 5846-5856 is semi-randomly selected in the 'user' range of the port space as defined by ICANN/IANA. This range is currently unassigned by IANA.

Free End-point Port Discovery

On startup each service publisher must attempt to connect to the first port in the port sequence. It will use the underlying OS and network stack to attempt to bind to this port.

All network access must go through the normal underlying OS mechanism. One service publisher must not block another publisher from accessing the network.

If the underlying OS reports that the port is already in use the service publisher will repeat the same process with the next port in the port sequence. This will be repeated until a port is successfully bound to, or all ports in the sequence have been tried.

If no available port can be found the service publisher will have failed to start. How this failure is handled by the service publisher is undefined.

It's important that a single organisation doesn't use up multiple ports, since this could lead to all the ports being blocked so that other publishers can't get a free port.

Any single organisation will publish all services on a single port, determined dynamically as above.

Note: *A service publisher will only fail to find a free port if more than 12 different hardware vendors are attempting to publish services from the same machine. This should be unusual.*

Handling Incoming Connections

Once a service publisher has successfully bound to a port it must handle connection attempts. It will accept all connections from any clients without filtering attempts. Security around connections will be handled after a connection has been established.

Note: *This document does not cover restrictions on connections to services or managing permissions for connections, such as limiting connections to certain machines or sub-nets. This would normally be under the control of the machine deployer and can be controlled through normal firewall settings and network configuration.*

Incoming connection attempts will specify a specific URI using the normal WebSocket process. The service publisher will allow connections to valid URIs as defined in this spec and track which URI each connection was made to.

The initial connection will be to the URI `wss://machinename:port/xfs4iot/v1.0`. This connection will then be used to list/discover individual services using the process outlined below ([Service discovery](#)).

[1.2.1.7 - Client](#)

A client application must be able to discover and open a connection to each service that it will use. It does this in two steps; firstly, through publisher end-point discovery, then through service discovery for each service end-point. It will do this through the following process.

[Publisher End-point Discovery](#)

The client will enumerate end-points by attempting to open a WebSocket connection to the following URL on each port in the port sequence. (See [Port sequence](#)). `wss://machinename:port/xfs4iot/v1.0`

The client will continue to enumerate publisher end-points by repeating for each port number in the port sequence until all ports have been tried.

The client will also start [service discovery](#) on the open connection. There is no requirement for the order of opening ports and discovering services. All ports connections may be created first followed by service discover, or port enumeration and service discovery may continue in parallel.

If the connection attempt to any port fails then the application will attempt error handling for network issues, machine powered off etc. The details of error handling are left up to the application.

[1.2.1.8 - Service Endpoint Discovery](#)

Once a connection has been established between the client and each publisher end-point, the client will discover the services published by sending a service discovery command and receiving events in the usual way.

The only command sent to the publisher end-point will be "Common.GetService".

```
{  
  "header": {  
    "type": "command",  
    "name": "Common.GetServices",  
    "requestId": "123"  
  },  
  "payload": {}  
}
```

The end-point will acknowledge the command in the normal way.

The command will be followed by zero or more events. The command will complete with a completion event, in the normal way. Each event, and the completion event will contain the following fields:

```
{  
  "header": {  
    "type": "completion",  
    "name": "Common.GetServices",  
    "requestId": "123"  
  },  
  "payload": {  
    "vendorName": "<Name of hardware/software vendor>",  
    "services": [  
      {  
        "serviceURI": "wss://machinename:port/xfs4iot/v1.0/<servicename1>"  
      },  
      {  
        "serviceURI": "wss://machinename:port/xfs4iot/v1.0/<servicename2>"  
      }  
    ]  
  }  
}
```

The service end-point URI will be returned as serviceURI.

A secure wss:// protocol URI will be returned whenever possible.

An insecure ws:// protocol URI may be returned instead. If an insecure ws:// protocol is used then the hardware vendor will be responsible for ensuring the security of the connection.

Much of the security of the XFS4IoT specification is based around TLS encryption. Using an unencrypted ws:// protocol will have a negative impact on that security, so as far as possible a wss:// should always be used.

If an unencrypted ws:// connection is used then alternative methods should be used to keep the connection secure, perhaps by physically securing the connection.

The Publisher service will send an event to report on every URI. A single event may report on one or more URI. URI will not be repeated between events, so each URI will be reported exactly once.

A publisher service may be designed to send one URI per event, or it may group URI together into a smaller number of events. The publisher should try and send events to report on each URI as soon as each URI is known. It's possible a publisher will know the complete set of URI when they're requested and can send them all at once in one or more events. Alternatively the URI may not be known straight away (such as if an IP address or port is being dynamically allocated.) In that case the publisher service would delay sending events for unknown URI until the full URI is known.

Having each URI reported at most once means that a client can connect to each URI reported in events without having to track which URI have already been connected to. This simplifies the client. Alternatively, a client may wait for the completion event and a full set of URI before attempting to connect. This would be simpler to implement, but might be slower to start up.

The completion event will contain every URI that the publisher service is aware of.

The publisher service will follow the above process to publish all URI that it's aware of. It will not suppress URI based on device status or service status.

For example, a device might be powered off, in the process of powering on, or powered on but have a hardware fault that makes it impossible to use. In all cases the publisher service will publish the URI anyway. The client can't assume anything about the device based on the URI. It will always need to query the service at the URI for its status to know more.

Events should be sent as soon as a URI is known by the publisher - the event doesn't mean or imply that the URI is currently available or can be connected to - that error handling must be performed by the client (see below.)

Note: *Even if the publisher service could know that a URI was valid at the time that it sends the event, the client can't know that the URI is still valid when it attempts to use the URI. It could have failed between querying and connecting. So the client has to handle errors, timeouts and retrying when connecting to the URI.*

The client may then attempt to open a WebSocket connection to each of the returned URI. The client will handle connection failures and timeouts by repeating the attempts to connect such that the service has a reasonable amount of time to start up.

Each service will endeavor to accept connections as quickly as possible during startup and restarts. Once a connection has been accepted a service may continue to report a 'starting' status until the device is physically started and ready.

Some devices are slow physically to start up, but software should be able to start relatively quickly. So, for example, a cash recycler device might be able to accept a connection within a few seconds of power being applied, but the physical hardware can take several minutes

to reset. During this time the service would accept connections but report a 'starting' status.

Each connection will be used to communicate with a single service. The service will then be queried for details about that service, such as the type of service or device that it represents and the messages and interfaces that it supports. (**todo: Querying for service information needs to be documented elsewhere.**)

The connection to the service will be kept open for as long as the service is in use. Details of the service lifetime are covered elsewhere.

The returned URI is a full URI including the machine name and port. It is possible that these values will be different to the service discovery URI - each service may be on a different machine, a different IP address, and a different port. The port is also independent of the discovery port range. It can be any port number.

The service URI values will have the same version number as the service discovery URI version number. Different versions of the API will not be mixed.

If a client wants to open multiple different API version numbers then it should perform service discovery against each of the possible version URI strings.

The client may close the publisher connection once it has completed service discovery, or it may keep the connection open. This will have no effect on the behavior of services.

1.2.2 - Messages

Overview of the general message handling process for XFS4IoT.

1.2.2.1 - API Definition

In XFS4IoT Services are accessible through a WebSocket Interface. The following specification details the format of message sent to the XFS4IoT Service.

Data sent across the WebSocket stream utilize JSON as a format (<https://www.json.org/>). Each message conforms to one of the following Message Types. Message Types are documented in the following table.

| Message Type | Direction | Description |
|--------------|-------------------|---|
| Command | Client to Service | Message sent to the XFS4IoT Service to perform a Command |
| Acknowledge | Service to Client | Message from the XFS4IoT Service indicating if the Command is valid and queued. |
| Event | Service to | Intermediate message from the XFS4IoT Service indicating |

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

| | | |
|-------------|-------------------|--|
| | Client | progress of the Command. |
| Completion | Service to Client | Final message from the XFS4IoT Service indicating the Command is complete. |
| Unsolicited | Service to Client | Message from the XFS4IoT Service unrelated to a Command. |

All the message types follow the same JSON structure conforming of a mandatory header and payload.

- **header** : containing attributes that are common across all Message Types to allow the payload to be efficiently parsed
- **payload** : containing information that is specific to the Message Type and action.

Header and Payload are the only two attributes defined at the top level of the JSON structure as the example illustrated below.

```
{
  "header": {
  },
  "payload": {
  }
}
```

1.2.2.2 - Header Definition

Headers are consistent across all XFS4IoT Message Types. All of the following attributes are mandatory.

- "type", string : The message type. This must be one of "command", "acknowledge", "event", "completion" or "unsolicited"
- "name", string : The message name, for example "CardReader.Status"
- "requestId", int : Unique request identifier supplied by the client used to correlate the command with acknowledge, event and completion messages. For unsolicited messages the value will be zero. For all other message types this will be an integer. The client will supply values that are positive, incremental and start with 1, so that unsolicited events can be distinguished. The service will check that the requestId does not conflict with a currently executing or queued command request from the same client and reject if it does.

The following example illustrates the header for a CardReader.Status command Message.

```
{  
  "header": {  
    "type": "command",  
    "name": "CardReader.Status",  
    "requestId": 12345  
  },  
  "payload": {}  
}
```

1.2.2.3 - Payload Definition

The XFS4IoT interface specifications detail the payload content for the class command, event, completion and unsolicited messages.

1.2.2.4 - Extra Fields

Since the XFS4IOT message format it JSON it is possible to include extra values not defined by the specification. This can be useful in some cases and is allowed as long as those extra values don't impact the proper functioning of the service or client.

For example, it may be useful to include fields with extra debugging information such as human readable error messages or hardware specific error codes.

When non-standard fields are used there's a risk that the same name could be used by different implementations, causing unexpected behaviours. Implementors should reduce the risk of this by using a company name or code as a prefix for the field name. For example, a company called "Acme" might add fields for a hardware error code and a log message. Good names for these would be "AcmeHardwareError" and "AcmeLogMessage", reducing the risk of the same name being used by a different implementation.

Any extra field not defined by this specification and not recognised by the Service or the Client will be ignored.

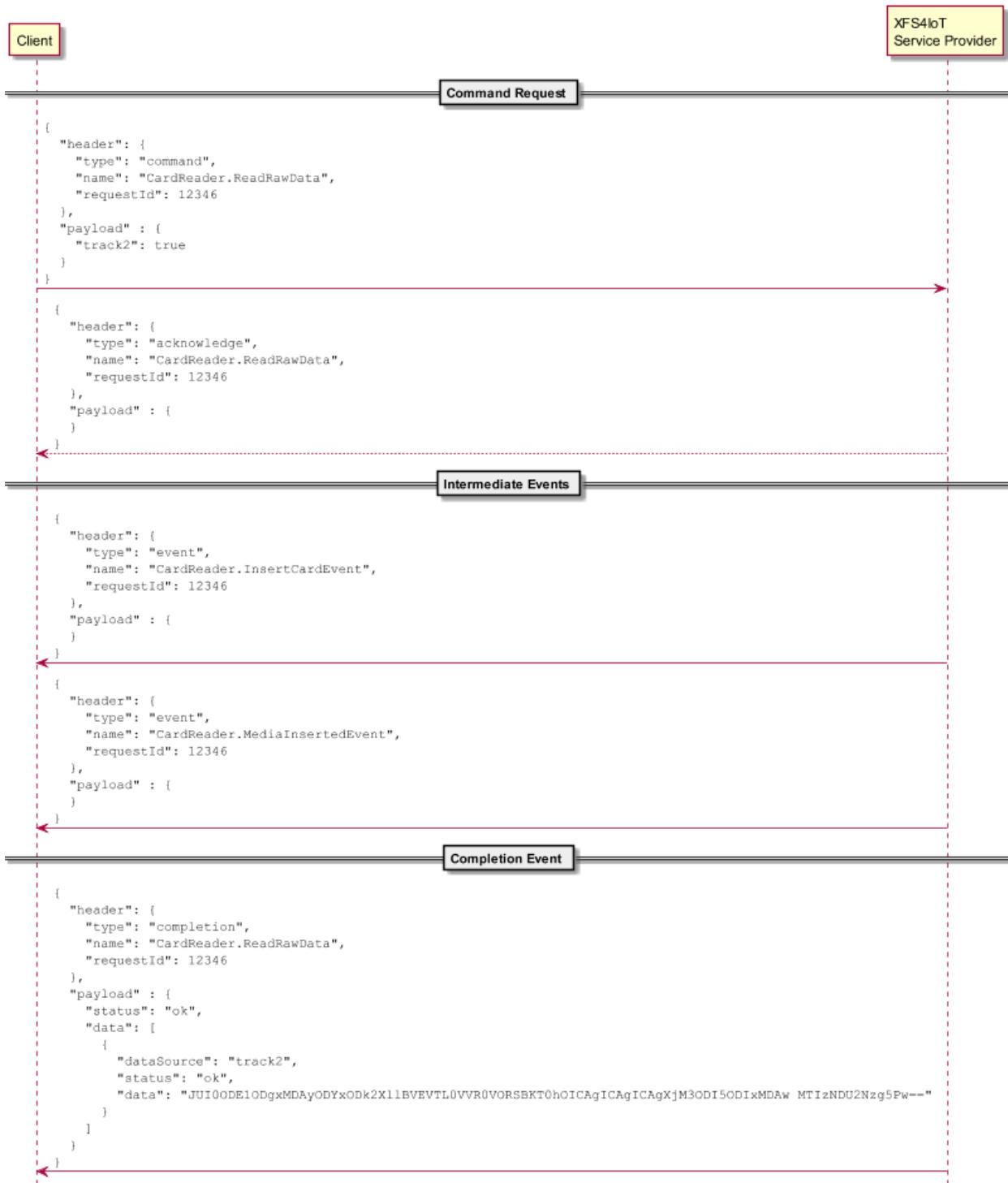
Ignoring an unknown field will have no effect on the standard behaviour of the service or client. There will be no requirement to use undefined fields.

The service or client may use undefined fields for whatever purpose they require. Dependence on undefined fields will mean the client or service is non-standard and may impact interoperability.

Implementers should pick undefined field names to avoid name clashes.

Note: work-in-progress. Use at your own risk.

1.2.2.5 - Example Command Request Message Sequence



1.2.3 - Command Sequence

Overview of the general sequencing of XFS4IoT command messages.

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

1.2.3.1 - *Introduction*

Once a service has been discovered and a connection created the client can send command messages to the service. Commands may cause the service to perform actions that are entirely software based, such as returning the current status, or they may cause actions to be performed by hardware, such as opening a shutter.

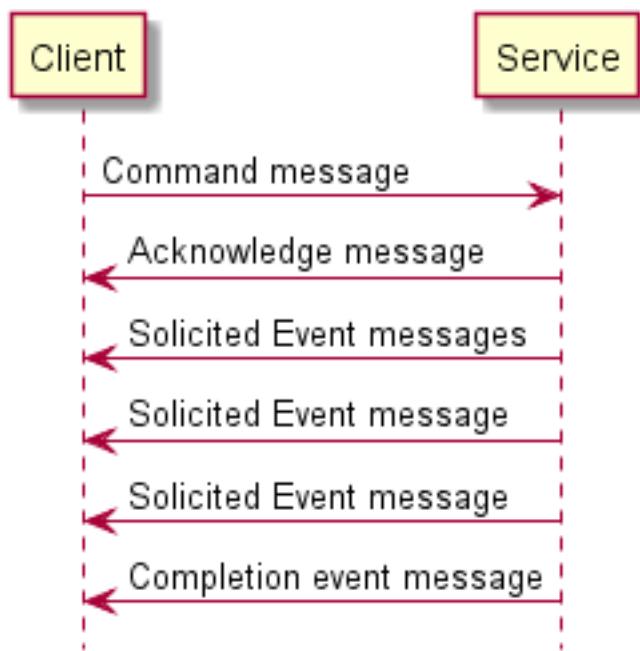
The sequence of messages passed between the service and the client is the same for all commands, independent of the command or interface being used.

Services may also send unsolicited events directly to the client. This can happen at any time that the service connection is open. This could be during the processing of a command, or between commands.

Unsolicited events may be sent at any time that there is a connection open between the service and the client.

1.2.3.2 - *Message Types*

The normal command message sequence will be:



All parts will be passed as standard messages as defined in the Messages section.

1.2.3.3 - Command Messages

The start of a command will be initiated by the client with a 'Command' message, requesting the service performs the specified action. The message use the standard header fields with:

- Type will be "command"
- Name will be the string defining the command to be executed as specified in the relevant interface specification.
- requestId will be a unique ID which is used to link future events to this invocation of the command.

The request ID given client and allows the client to link messages sent in response to the command back to the original command. For example, the completion event for this command will contain the same request id.

The value of the request IDs will be a unsigned integer, incremented between each command. The client is responsible for ensuring that each request ID is unique for a single connection. (Note that request ID's don't have to be unique across connections. The request is identified by a combination of the request ID and the connection.)

The request ID value 'zero' will be used for unsolic events, so the client should create request IDs starting with one or higher.

The request ID will be created by the client. It will be a unsigned integer strictly greater than zero, and will be incremented for each request.

The service will remember the last request ID and reject any request ID for a new command which is lower or equal to the previous request ID. Other than that the service will not track the request ID.

1.2.3.4 - Acknowledge Messages

As soon as the service has received and decoded the command message it will send an acknowledge message to indicate that it has the message. This will normally include the request ID so that the client can identify which command it relates to (unless there's some sort of error that stops the request ID being included).

Sending the acknowledge message immediately allows the client to handle network errors and lost messages more quickly. It can set a short timeout and expect to receive the acknowledge within that timeout, and continue with error handling if it doesn't.

Receiving the acknowledge doesn't give any guarantees about what the service will do with the command, or even that it can be executed. Any errors will be reported in the completion event for the command, not in the acknowledge.

The only exception is when it would be impossible for the service to send a useful completion event, such as if there's no request ID to include in the completion event. In this small number of cases an error code will be included in the acknowledge message.

Errors will include:

- invalidMessage : Message can't be decoded
- invalidRequestID : request ID is not an unsigned integer, or not greater than previous request IDs.
- tooManyRequests : There are currently too many requests queued by the service and the service can't queue any more.

Note: *The error codes are intentionally defined to be very simple and not cover all cases. Extra information about exactly what caused a failure can be included in non-standard debugging fields in the acknowledge message.*

[1.2.3.5 - Event messages](#)

During the processing of the command the service can send multiple solicited events, as defined in the interface documents. This is used to inform the client when something significant happens that it may need to react to, like a card being inserted or a key being pressed.

Each solicited event will contain the original request ID in the header, and will only be sent on the connection that the original command was received on, so that individual solicited events can be linked to the original command by the client.

For compatibility with future specification changes, and to permit custom extensions by service implementors, the client should ignore any events that it doesn't recognise.

The client will ignore any unknown events.

[1.2.3.6 - Completion Messages](#)

The normal processing of commands will complete with exactly one completion event. The exact type of the completion event will depend on the command and is defined in the interface specification for that command.

Exactly one completion event will be sent by the service for each executed command message.

If an acknowledge message with an error code is returned to the command message then the command will not be executed, and no completion even will be sent.

The completion event will contain the request ID from the original command message, so that the client can link the message back to the command. Once the completion event has been sent, that command handing has completed and no more messages will be sent for it.

After the completion event for a command message, with a particular request ID, has been sent then no more events will be sent with that request ID.

Each completion event will contain as much information as possible to avoid requiring extra events. For example, when a command is used to fetch information from the service then the information will be included in the completion event. When a command results in particular information, like reading a card, then that information is included in the completion event. The exact information included in each completion event is defined in the interface document that defines that completion event.

1.2.3.7 - Error Codes

Once a command is started, after the acknowledge message has been received, any error will be included in the completion event for that command. The interface specification will define which errors are valid for each completion event - no other errors will be returned by a service.

Common error codes are defined in the completionCode property of the completion message. Extra error properties may be defined in the interface definition for each command. Extra information about the details of the error may be included alongside the main error code, which may be used by the client in some cases, or may be used for diagnostics and logging.

1.2.3.8 - Unsolicited Event Messages

Service will also send 'unsolicited' events to the client to signal events that can happen at any time, independent of command handling. These can happen before, during, or after any command handling.

To allow clients to react to events quickly, unsolicited messages should be sent as soon as possible. As far as possible the service will send unsolicited events immediately. For example, it should avoid queueing events until after the current command has been processed if it doesn't have to.

Since unsolicited events aren't linked to command handling, they do not have a matching request ID. The event header will contain a request ID field (since all events do,) but the request ID will always be zero for unsolicited events. Unsolicited events are also broadcast to all clients, on all open connections.

unsolicited events will have a request ID of zero.

unsolicited events will be sent to all open connections.

The valid unsolicited events for each interface will be defined in the relevant documentation.

For compatibility with future specification changes, and to permit custom extensions by service implementors, the client should ignore any events that it doesn't recognise.

The client will ignore any unknown events.

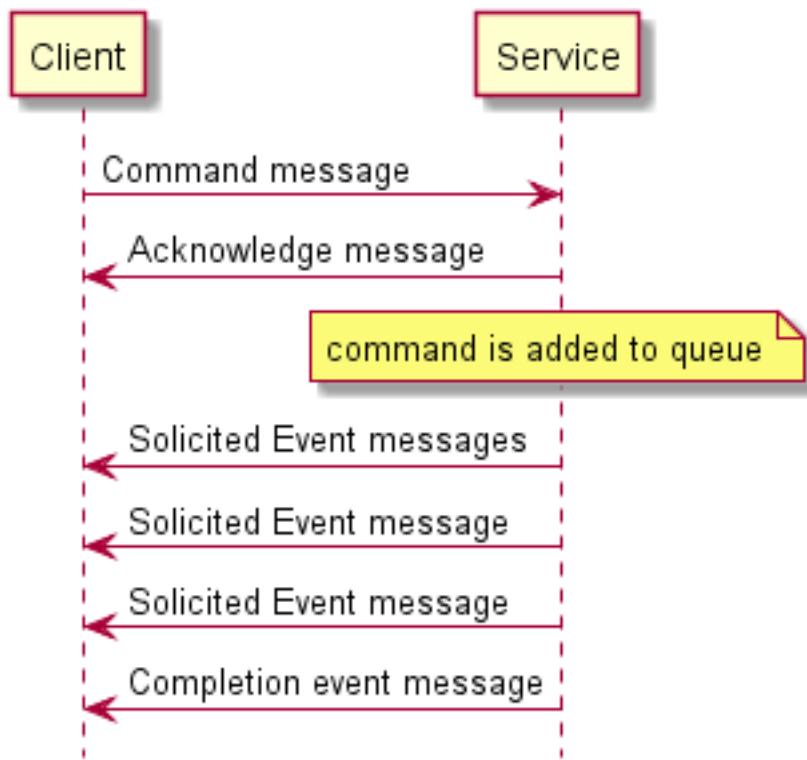
1.2.3.9 - Command Queueing

Some commands can be executed in parallel. For example, a status command that returns the current status can always be executed immediately even if another long running command is being executed. Other commands may be blocked from parallel execution by mechanical or other restraints. For example, it's probably impossible to accept a card and capture a card at the same time on most card readers.

As far as possible services will attempt to execute commands in parallel. In particular, all commands that simply return information should be executed immediately even if other commands are in progress. It is up to the client to synchronise status information with ongoing actions.

When it's not possible to execute a command immediately, possibly because a mechanical operation is already using the hardware, then commands will be queued up and executed as soon as possible.

The acknowledge message is never queued.



When a command can not be executed immediately it will be queued until it can be executed.

Queued commands will be executed in the order they are received.

1.2.3.10 - Cancellation

A client can send a cancel command for an existing command at any time.

The cancel command will be a new command, with it's own request ID and it's own completion event.

The request ID of the command to cancel will be included as a parameter of the cancel command, allowing the service to identify the command to cancel.

The cancel command will complete straight away. It will not wait until the original command has been completed or cancelled. The cancel command will normally complete as "successful" even if the original command hasn't or can't be cancelled.

If the request ID of the original command is lower than any known request IDs then the cancel command will complete as "successful". This means that a cancel for a command that has already completed will always complete as "successful". The service won't make any other attempt to validate that the request ID was valid. The service will not track completed commands.

The cancel command will complete with an error if the request ID for the original command is invalid - that is, it's higher than any existing request ID.

The cancel command itself can not be cancelled. Any attempt to cancel a request ID for a cancel command will complete with an error.

Service will make a best attempt to cancel the command as soon as possible. However it may not always be possible to cancel a command immediately, or at all.

If a command can't be cancelled it will continue and complete as if no cancel message had been sent. All commands will end with exactly one completion message; either a normal completion message or a cancel completion message.

A cancelled completion message will be handled exactly like any other completion message. For example, there will be no more solicited events sent after the cancelled completion.

[**1.2.4 - End to End Security**](#)

Overview of the general end to end security support in XFS4IoT.

[**1.2.4.1 - Overview**](#)

A key priority for XFS4IoT is to improve security of the entire environment where XFS is used. This means securing not only the interface between the service and the hardware, or the interface between the client and the service, but providing security all the way from one end of an operation to the other.

For example, during a cash dispense operation the transaction will first be authorised by an authorising host which represents the owner of the cash in the device. That host will communicate through various other systems to the client application, the client application will communicate with the XFS4IoT service and the service will finally communicate with the hardware. Any part of that process is vulnerable to an attack which could lead to the wrong amount of cash being dispensed. XFS4IoT has been designed to block attacks at any point between the authorising host and the dispenser hardware - it is truly end to end.

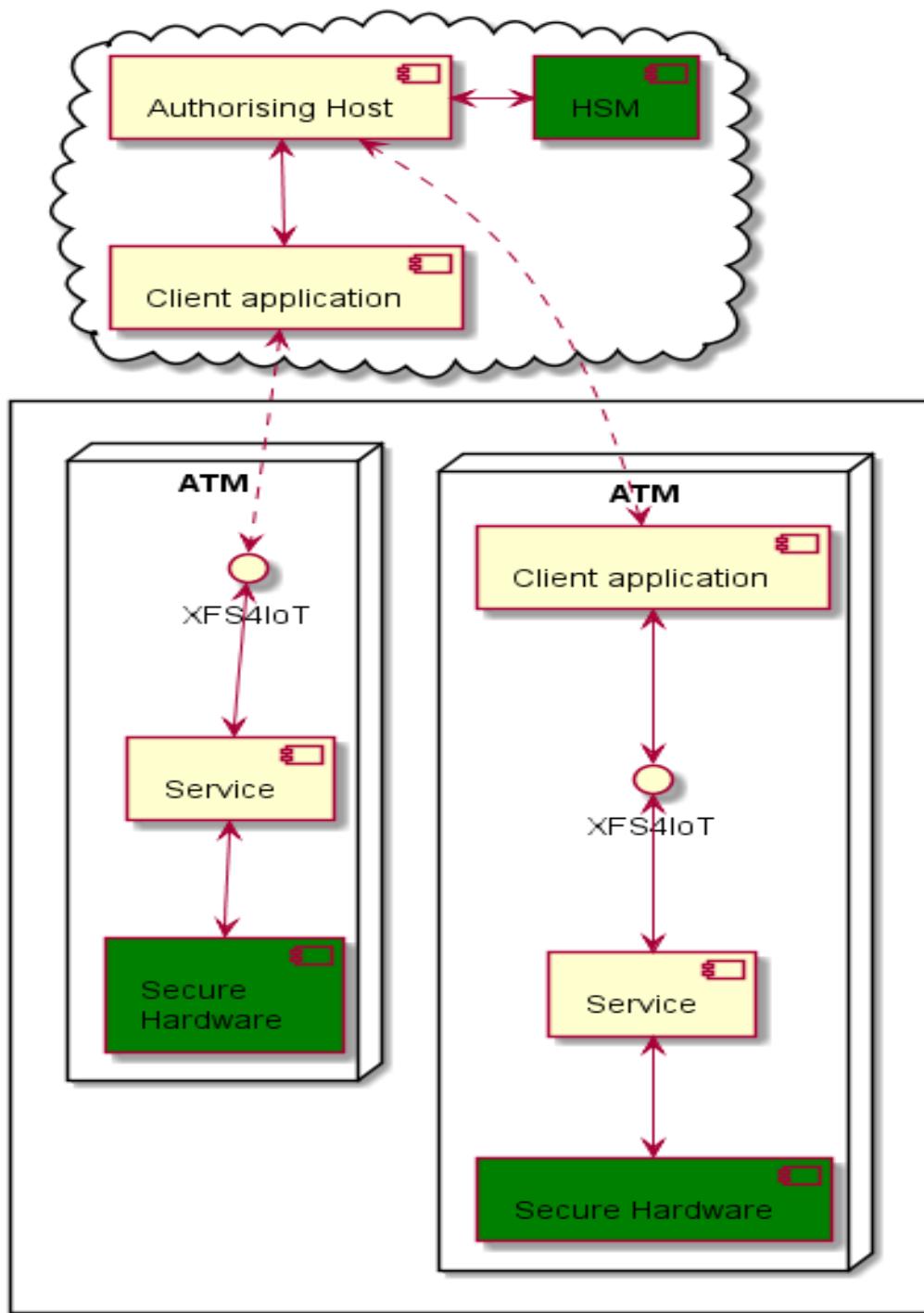
Both data 'integrity' like this, and 'confidentiality' for things like card data, are important. XFS4IoT focuses on integrity since confidentiality is covered by TLS encryption of the network messages.

[**1.2.4.2 - General end to end sequence**](#)

End to end security involves communicating facts between two end points, and validating that those facts have not been changed or tampered with. Typically this would be between a Hardware Security Module (HSM) in a data-center, and a Hardware Security Element

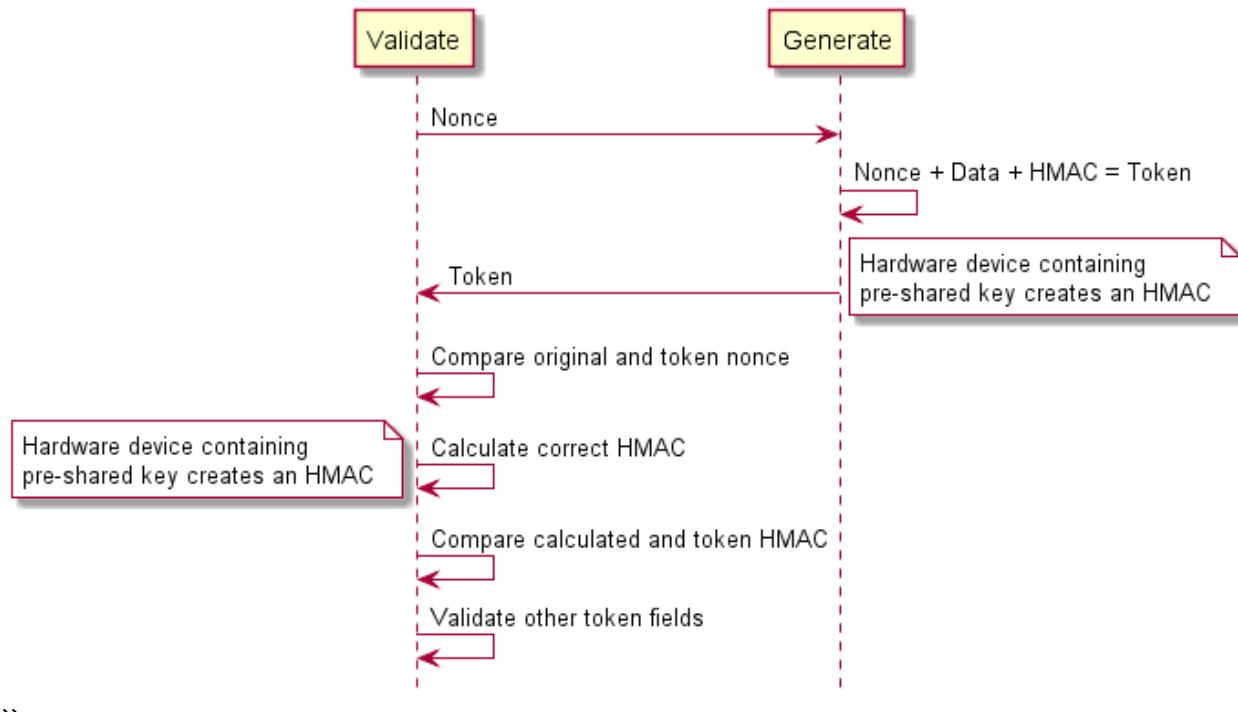
(HSE) built into a device such as a cash dispenser. The method of communication between these two ends does not matter since any tampering with the data will be detected.

The following diagram shows a typical system with the secure hardware end points in green. It shows two possible architectures, one with the client application running locally on the ATM, and the other with application running in the cloud.



The general sequence for an end to end security operation involves one system generating a token, and another system validating that the token has not been changed. This sequence looks roughly the same in both directions; for example an Authorising Host can create a 'Dispense Token' and the Dispenser will validate that token before dispensing cash. In the opposite direction, the dispenser can create a 'Present Status' token to protect information about the presented cash and the client can validate that the token is valid and has not been tampered with.

In both directions the sequence looks like this:



1.2.4.3 - Tokens

XFS4IoT implements end to end security using strong cryptography, such as HMAC values. Secret keys are shared between endpoints, ideally by using TR34 and TR31. The use of public key RKL ensures that only the correct endpoints have access to the end to end transactions.

The data relevant to the end to end transactions is separated out from the normal clear text properties passed in the JSON message formats, and is included in different "token" properties in the messages. Each token property is defined in the relevant interface documentation. For example, there is a "dispense token" for cash dispense actions and a "present status" token to protect the information about the last presented cash operation. These are defined in Dispenser service class documentation.

Tokens are encoded as a simple string which contains all of the information needed for that token, and also the information needed to keep it secure such as the HMAC value. Keeping all of the information together in this way ensures that a single HMAC can protect all of the data, and there is no possibility that part of the data could be changed.

Keeping the token as a simple string means that it is easy to handle for low-power hardware. For example, the token may need to be checked by embedded firmware, even if the service is running on a front end machine. To be fully end to end it must be checked on the hardware, so the format is kept simple.

The general format of a token is a string with a set of key=value pairs all comma separated and UTC8 encoded. The first pair will give the nonce. The last value will be an SHA256 HMAC, The UTC8 encoding is important so that the HMAC is consistent.

There will be multiple other key=value pairs, separated by commas. Keys can appear in any order. Key names are always upper case.

Tokens will not contain any extra whitespace.

To avoid possible parsing errors, binary data in tokens including the HMAC is encoded as simple hex values, rather than the normal BASE64 encoding. This is because BASE64 uses the "=" character which could be confused for the KEY/Value separator. Hex encoded data will be all upper case.

For similar reasons, the characters "=" and "," must never be used in any value data, including custom values. If custom key/value pairs are used then care must be taken that the value never contains those two characters.

The KEY=value pairs define what the token is used for. For example, the Dispenser service class has "DISPENSE1" and "PRESENTSTATUS" keys in different Tokens. The different key names ensures that tokens with different uses can not be reused by an attacker. "value" is the actually value of the data being protected and will be different for each operation.

NONCE is always the first key. HMAC256 is always the last key.

The set of standard key names and format for each value is defined in the relevant specification for each token type, including which keys are required and which are optional.

It is also permitted to include custom key values in a token. For example, a hardware dependent error code might be included. Unknown keys will be included in the HMAC calculation, but otherwise ignored. There must not be a dependency on custom keys. Care should be taken to avoid name clashes between keys, maybe by using a vendors name in the key name. For example, if the Acme Corporation wants to include "ERRORCODE" as a custom key name then they should call it something like "ACMEERRORCODE" and *not* "ERRORCODE".

The total token length will be limited to 1024 bytes to avoid the risk of buffer overflows. Any token longer than this will be treated as invalid data. The limit is in bytes not characters since UTC8 characters may include multiple bytes.

1.2.4.4 - Token Keys

End to End security tokens are made up of key/value pairs. There are various key names that are common to all types of tokens, plus key names which are only valid for specific token types. The following key names are valid for all tokens.

- **NONCE** : The token nonce value, which was initially exchanged between the end points and is used to ensure that every token is different and to avoid replay attacks. The nonce will always come at the start of the token (so that there is not too much constant leading data - this is important for security.)

The nonce value must be meet the cryptographic requirements for a nonce. It must be *non-repeating* - that is, the same nonce value must never be used twice. Code creating a nonce value should be carefully reviewed to make sure this is true.

There are two simply ways of guaranteeing this:

An incrementing integer can be used, as long as it always increments between every token. This must be true across restarts and power fails. The counter *must not* reset and repeat the same values. (Note that it doesn't actually matter that the value can be predicted.)

A *strong* random number can be used as long as the random number never repeats. This can be easy if there is a hardware random number generator available. If a pseudo-random number is used then the seed value needs to be carefully picked so that it never repeats (which would cause the random number to repeat.) Needing to track unique seed values might mean it's easier to simply use a persistent integer counter.

- **TOKENFORMAT** : The version number of the token format. Currently this will always be "1"
- **TOKENLENGTH** : The total number of bytes in the token, including the HMAC value, in decimal. This value will be exactly four digits and include leading zero's as required. Since this value has to include the length of itself, making it fixed length makes it easier to calculate. Note that this is *bytes* and not characters, since UTC8 characters may contain multiple bytes.
- **HMACSHA256** : The HEX encoded HMAC of all the preceding data up to and including the last equals after the HMACSHA256 key. The HMAC is always the last value - this makes it easy to calculate the HMAC since it can be calculated over all other data, converted to hex, then appended to the string. The HMAC will use SHA256 as the algorithm. The hex encoded data will be all upper case.
- Other KEY=value pairs define what the token is used for. For example, the Dispenser service class has "DISPENSE1" and "PRESENTSTATUS" keys in different Tokens. The different key names ensures that tokens with different uses can not be reused by an attacker. "value" is the actually value of the data being protected and will be different for each operation.

1.2.4.5 - Token Examples

The general token format looks something like this:

```
NONCE=<nonce  
value>,TOKENFORMAT=1,TOKENLENGTH=<Length>,<KEY>=<value>,HMACSHA256  
=<HMAC>
```

For example, a dispense token might be:

```
NONCE=254611E63B2531576314E86527338D61,TOKENFORMAT=1,TOKENLENGT  
H=0164,DISPENSE1=50.00EUR,HMACSHA256=CB735612FD6141213C2827FB5A6A  
4F4846D7A7347B15434916FEA6AC16F3D2F2
```

The value used to calculate the HMAC is

```
NONCE=254611E63B2531576314E86527338D61,TOKENFORMAT=1,TOKENLENGT  
H=0164,DISPENSE1=50.00EUR,HMACSHA256=
```

A HMAC for this data, with SHA256 and a key of
112233445566778899AABBCCDDEEFF112233445566778899AABBCCDDEEFF, is

```
CB735612FD6141213C2827FB5A6A4F4846D7A7347B15434916FEA6AC16F3D2F2
```

This would be included in a dispense command message as:

```
{  
  "header": {  
    "type": "command",  
    "name": "dispenser.dispense",  
    "requestId": 456  
  },  
  "payload": {  
    ...  
    "dispenseToken":  
      "NONCE=254611E63B2531576314E86527338D61,TOKENFORMAT=1,TOKENLENG  
      TH=0164,  
      DISPENSE1=50.00EUR,HMACSHA256=CB735612FD6141213C2827FB5A6A4F4846D  
      7A7347B15434916FEA6AC16F3D2F2"  
    ...  
  }  
}
```

Similarly the PresentStatus might include a PresentStatus token:

NONCE=1414,TOKENFORMAT=1,TOKENLENGTH=0268,DISPENSEID=CB735612FD6141213C2827FB5A6A4F4846D7A7347B15434916FEA6AC16F3D2F2,
DISPENSED1=50.00EUR,PRESENTED1=YES,PRESENTEDAMOUNT1=50.00EUR,RETRACTED1=NO,
HMACSHA256=55D123E9EE64F0CC3D1CD4F953348B441E521BBACCD6998C6F51D645D71E6C83

which would be included in the present status completion event as:

```
{  
  "header": {  
    "requestId": 765,  
    "type": "completion",  
    "name": "dispenser.presentStatus"  
  },  
  "payload": {  
    ...  
  
    "denomination": {  
      "currencies": [  
        {  
          "currencyID": "EUR",  
          "amount": 50  
        }  
      ],  
      ...  
    },  
    "presentState": "presented",  
    "token": "NONCE=1414,TOKENFORMAT=1,TOKENLENGTH=0268,  
    DISPENSEID=CB735612FD6141213C2827FB5A6A4F4846D7A7347B15434916FEA6AC16F3D2F2,DISPENSED1=50.00EUR,PRESENTED1=YES,  
    PRESENTEDAMOUNT1=50.00EUR,RETRACTED1=NO,HMACSHA256=55D123E9EE64F0CC3D1CD4F953348B441E521BBACCD6998C6F51D645D71E6C83"  
    }  
  }  
}
```

Note: token strings never have new lines or any white-space. New lines are included in examples only to make them more readable

In this example €50.00 was moved from the cassettes, possibly to a stacker, and €50.00 was then presented to the customer. The notes were not retracted so it should be assumed that the customer could have taken the notes.

1.2.4.6 - Encryption Key Management

To ensure strong security, secret symmetric encryption keys are shared between different endpoints.

The keys used for end to end encryption have fixed names. They are XFSAuthenticateHost and XFSAuthenticateDevice, where XFSAuthenticateHost is used by the host to create an HMAC, and XFSAuthenticateDevice is used by the hardware to create an HMAC. Both keys must be shared between both endpoints so that HMAC values can be both calculated and checked.

Ideally sharing the keys will be done using public key encryption, as defined in TR34. However, in some cases it may be necessary to pre-load keys into hardware in some other secure way, for example in legacy hardware that can not support TR34. The security of the whole system is only as good as the security of these keys so it is vital that this is done in as secure a way as possible.

The key details are defined by the TR31 specification which defines both key data and details such the intended use. This includes the algorithm that the key can be used with. Since TR31 defines the algorithm there is no need for algorithms to be negotiated in any other way - the two end points will effectively agree the algorithm to use by loading the relevant keys.

Encryption keys must be long enough to ensure security when used with a particular algorithm. For example, where SHA256 is used for the HMAC value the key must be at least 256 bits long to match the algorithm.

For simplicity, communication in each direction will be handled separately. So, for example, there will be a separate key for tokens passed in each direction. (Also there will be a separate nonce in each direction. See below.)

The details of the key management are covered by the shared key management interface. Any service that implements end to end security will implement this interface as part of its interface.

1.2.4.7 - Unique messages and replay attacks

To avoid 'replay attacks', where an attacker reuses an old message to replace a new one, it is important that all individual tokens are unique and the same data is not used multiple times. For example, it is common to have a dispense token for "10EUR" so that value (and its HMAC) will be the same for many transactions and could be reused by an attacker. To avoid this a "Nonce" value is included in each token. The nonce will be different for each transaction. This guarantees uniqueness.

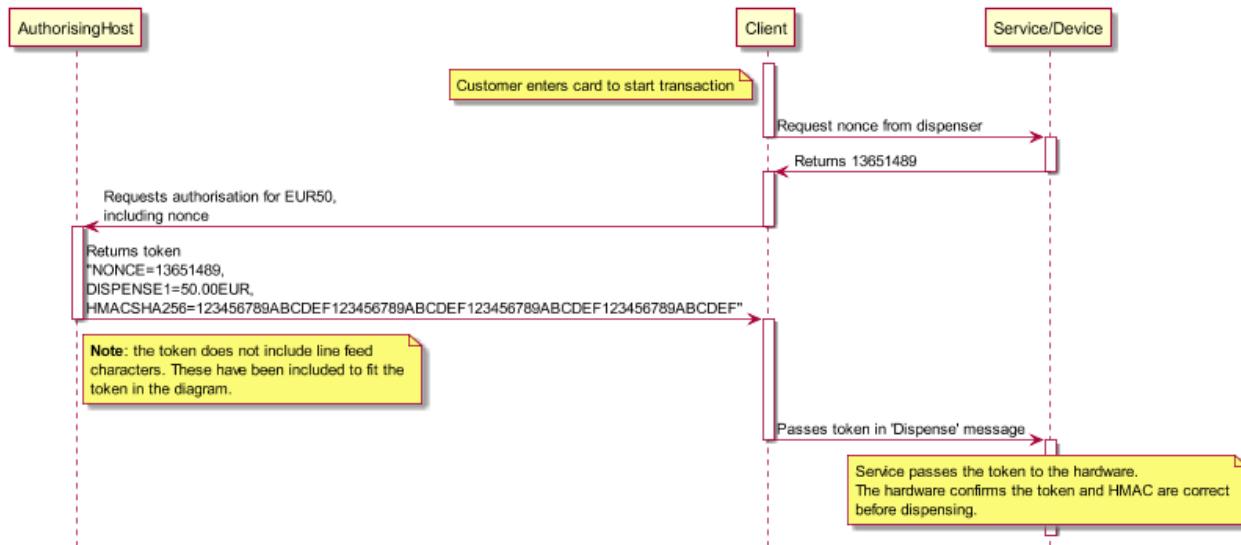
There will be a different nonce for tokens passed in each direction. That is, there will be a command nonce and response nonce. Each nonce must be generated and checked at the same end of the communication, so the command nonce must be generated and checked by

the device hardware. The response nonce must be generated and checked by the client. (Ideally in the host HSM.)

To fully avoid replay attacks the nonce must be agreed between the endpoints before it is used. An extra command on the interface is called by the client to fetch a new command nonce. This nonce is then remembered by both end points and included in each command token. Similarly, a response nonce must be generated by the client/host, passed to the service, and included in all response tokens. It should be possible to add the response nonce to existing messages, such as the PresentStatus command message for the cash dispenser.

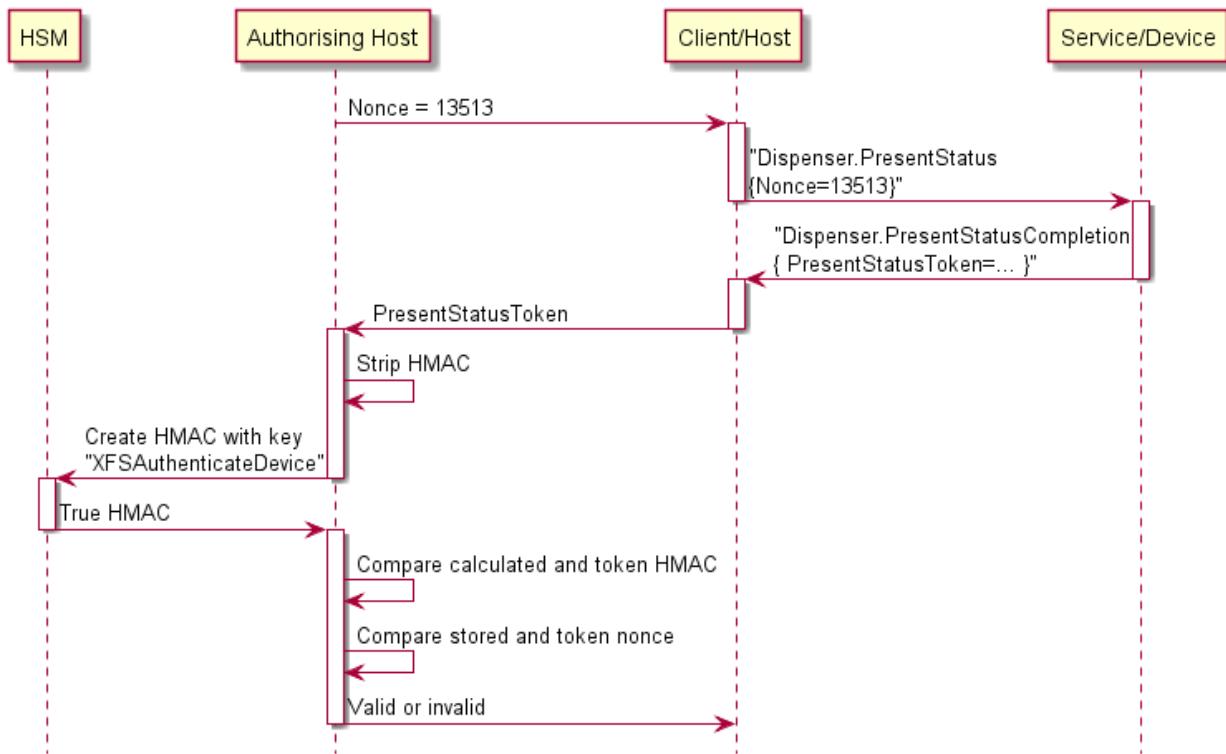
1.2.4.8 - Example: A classic dispense operation

The following example shows how end to end security is used to protect a cash dispense operation during a classic ATM transaction.



Once a dispense has been performed it is important to accurately report the result. If an attacker can change the reported result then them might fake an error to make it look like cash was not presented to the customer, and tricking the banks into reversing the transaction - this is known as transaction reversal fraud.

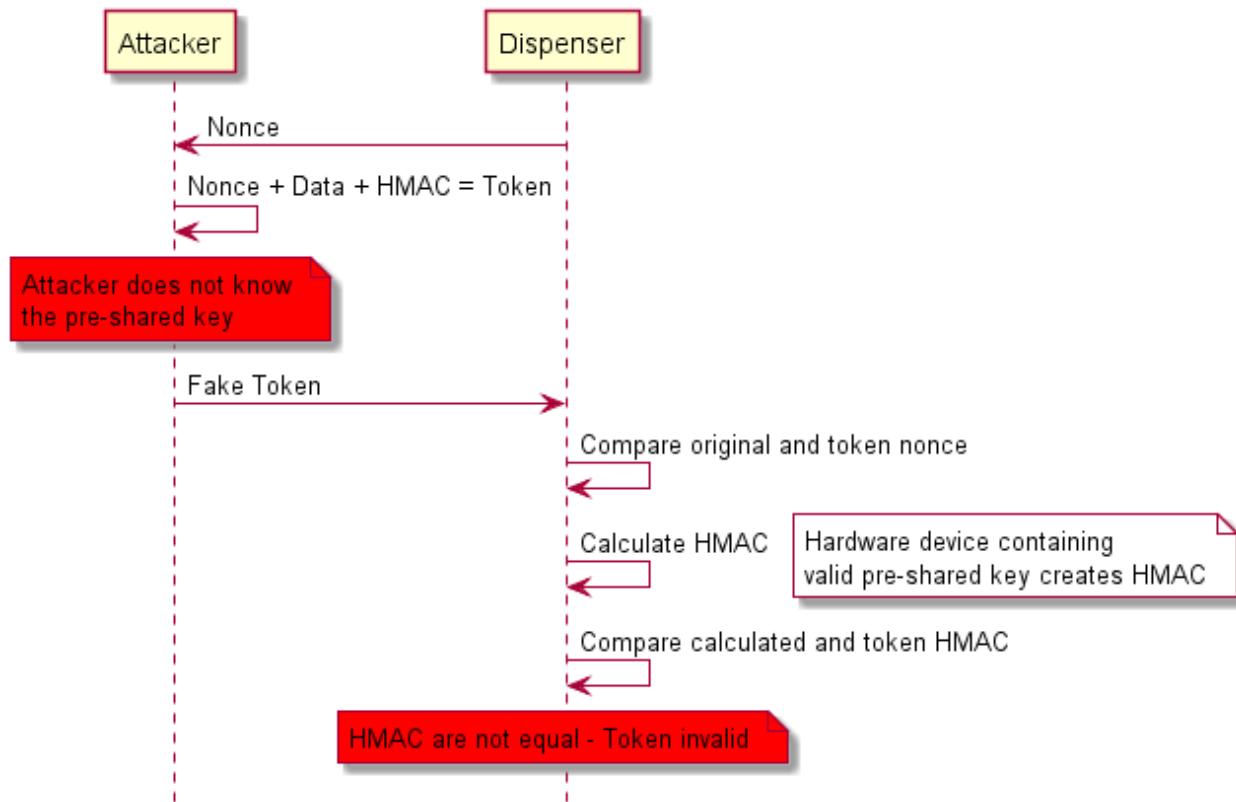
To protect against this, the PresentStatus command returns a token that can not be tampered with. Note that this token goes in the opposite direction, from the hardware to the host. This means that the nonce now coming from the host rather than from the hardware. The nonce can be included in the PresentStatus command without needing to call an extra command.



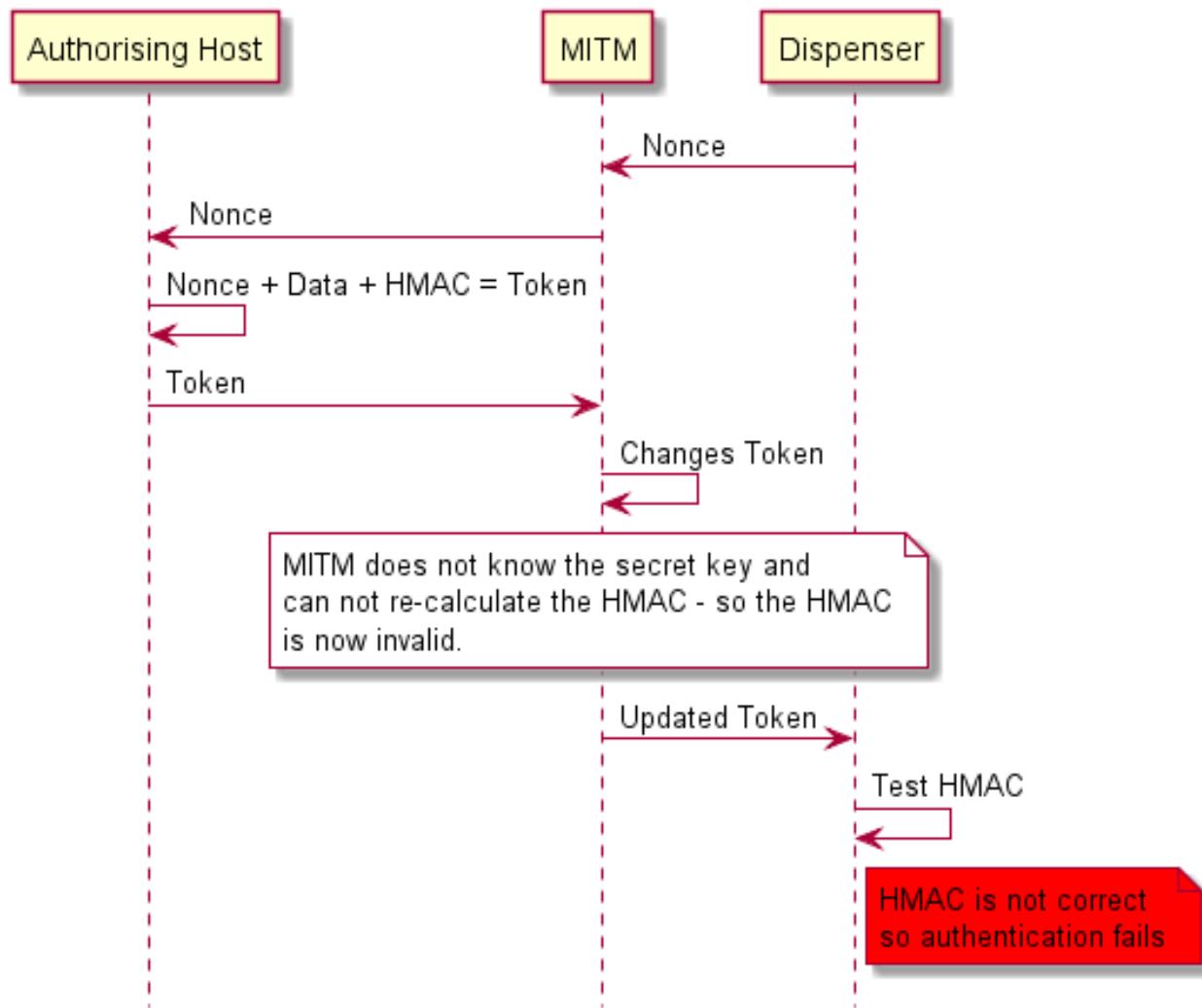
1.2.4.9 - Example: Types of attacks that are blocked

Various types of attacks are blocked by end to end security. Some examples are given here.

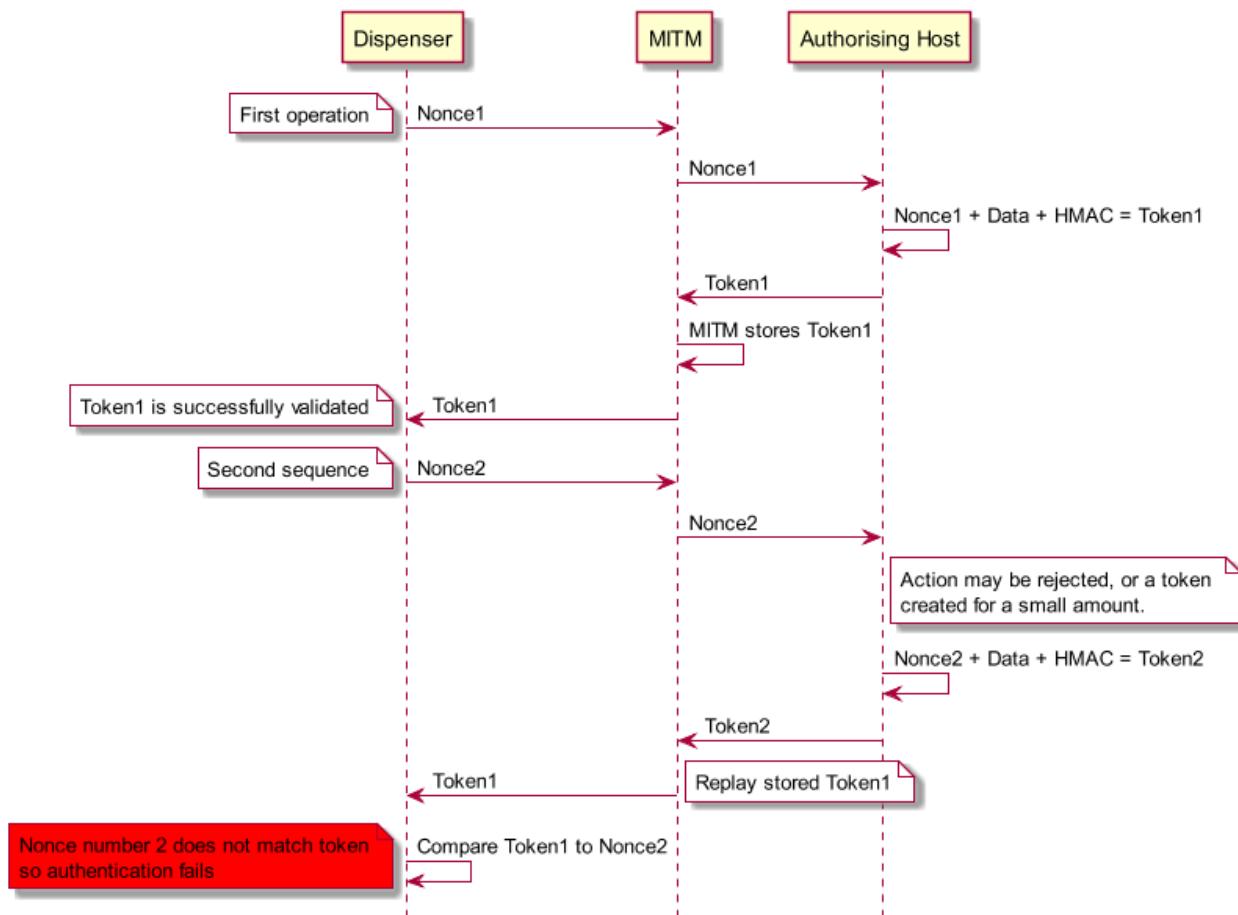
A fake client attempts to issue a dispense command without authorisation. This is commonly known as a "black box" attack.



An attacker with control over communications tries to change details, for example to increase the amount of cash being dispensed. This is known as a "Man in the Middle" attack.



An attacker with control over communication stores a token and attempts to use it a second time. This is known as a "replay attack"



1.3 - Command Messages

1.3.1 - Common.GetService

Command send to the service discovery port to find the details of the service exposed by this publisher

Command Message

| Payload | Type | Required |
|-----------------|---------|----------|
| { | | |
| "timeout": 5000 | integer | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "vendorname": Add example to YAML, | string | |
| "services": [{ | array (object) | |
| "serviceURI": wss://ATM1:123/xfs4iot/v1.0/CardReader | string | |

services/serviceURI

The URI which can be used to contact this individual service

Event Messages

- [Common.ServiceDetailEvent](#)

1.4 - Event Messages

1.4.1 - Common.ServiceDetailEvent

Details of some services published by this endpoint

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| " vendorname ": Add example to YAML, | string | |
| " services ": [{ | array (object) | |
| " serviceURI ": wss://ATM1:123/xfs4iot/v1.0/CardReader | string | |
| } | | |
| } | | |

Properties

vendorname

Freeform string naming the hardware vendor

services/serviceURI

The URI which can be used to contact this individual service

2 - Common Interface

This chapter defines the Common interface functionality and messages.

2.1 - Summary

TODO

2.2 - Command Messages

2.2.1 - Common.Status

This command is used to obtain the overall status of any XFS4IoT service. The status includes common status information and can include zero or more interface specific status objects, depending on the implemented interfaces of the service. It may also return vendor-specific status information.

Command Message

| Payload | Type | Required |
|---------------------------|---------|----------|
| { "timeout": 5000 } | integer | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|---------|------|----------|
|---------|------|----------|

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

```
{
  "common": {
    "device": "online",
    "extra": [Add example to YAML],
    "guideLights": [{},
      ],
    "devicePosition": "inposition",
    "powerSaveRecoveryTime": 0,
    "antiFraudModule": "notSupp"
  },
  "cardReader": {
    "media": "notSupported",
    "retainBin": "notSupported",
    "security": "notSupported",
    "numberCards": 0,
    "chipPower": "notSupported",
    "chipModule": "ok",
    "magWriteModule": "ok",
    "frontImageModule": "ok",
    "backImageModule": "ok",
    "parkingStationMedia": ["present","present"]
  },
  "cashAcceptor": {
    "intermediateStacker": "empty",
    "stackerItems": "customerAccess",
    "banknoteReader": "customokerAccess",
    "dropBox": false
  }
}
```

```

"positions": [
    "position": "inLeft",           array (object)
    "shutter": "closed",            string
    "positionStatus": "empty",      string
    "transport": "ok",              string
    "transportStatus": "empty",     string
    "jammedShutterPosition": "notSupported" string
],
"mixedMode": "notActive"          string
},
"cashDispenser": {
    "intermediateStacker": "empty", string
    "positions": [
        "position": "left",           array (object)
        "shutter": "closed",          string
        "positionStatus": "empty",    string
        "transport": "ok",            string
        "transportStatus": "empty",   string
        "jammedShutterPosition": "notSupported" string
    ]
},
"cashManagement": {
    "safeDoor": "doorNotSupported", string
    "dispenser": "ok",               string
    "acceptor": "ok"                string
}

```

```

"keyManagement": { object
  "encryptionState": "ready", string
  "certificateState": "unknown" string
},
"keyboard": { object
  "autoBeepMode": "active" string
},
"textTerminal": { object
  "keyboard": "on", string
  "keyLock": "on", string
  "displaySizeX": 0, integer
  "displaySizeY": 0, integer
  "leds": [{ array (object)
    "na": false, boolean
    "off": false, boolean
    "slowFlash": false, boolean
    "mediumFlash": false, boolean
    "quickFlash": false, boolean
    "continuous": false, boolean
    "red": false, boolean
    "green": false, boolean
    "yellow": false, boolean
    "blue": false, boolean
    "cyan": false, boolean
    "magenta": false, boolean
    "white": false, boolean
  }]
},
}

```

| | |
|-----------------------------|----------------|
| "printer": { | object |
| "media": "notSupported", | string |
| "paper": { | object |
| "upper": "notSupported", | string |
| "lower": "notSupported", | string |
| "external": "notSupported", | string |
| "aux": "notSupported", | string |
| "aux2": "notSupported", | string |
| "park": "notSupported" | string |
| }, | |
| "toner": "notSupported", | string |
| "ink": "notSupported", | string |
| "lamp": "notSupported", | string |
| "retractBins": [| array (object) |
| "state": "unknown", | string |
| "count": 0 | integer |
|], | |
| "mediaOnStacker": 0, | integer |
| "paperType": { | object |
| "upper": "unknown", | string |
| "lower": "unknown", | string |
| "external": "unknown", | string |
| "aux": "unknown", | string |
| "aux2": "unknown", | string |
| "park": "unknown" | string |
| }, | |

```

"blackMarkMode": "notSupported"           string
},
"cardEmbosser": {                         object
"media": "present",                      string
"retainBin": "ok",                        string
"outputBin": "ok",                        string
"inputBin": "ok",                         string
"totalCards": 0,                          integer
"outputCards": 0,                         integer
"retainCards": 0,                         integer
"toner": "full"                           string
},
"barcodeReader": {                         object
"scanner": "on"                           string
},
"biometric": {                            object
"subject": "present",                    string
"capture": false,                        boolean
"dataPersistence": {                     object
"persist": false,                        boolean
"clear": false                           boolean
},
"remainingStorage": Add example to YAML   string
}
}

```

Properties

common

Status information common to all XFS4IoT services.

common/device

Specifies the state of the device.

common/extra

Specifies a list of vendor-specific, or any other extended, information. The information is returned as a series of "key=value" strings so that it is easily extendable by Service Providers.

common/guideLights

Specifies the state of the guidance light indicators. A number of guidance light types are defined below. Vendor specific guidance lights are defined starting from the end of the array.

common/devicePosition

Position of the device.

common/powerSaveRecoveryTime

Specifies the actual number of seconds required by the device to resume its normal operational state from the current power saving mode. This value is zero if either the power saving mode has not been activated or no power save control is supported

common/antiFraudModule

Specifies the state of the anti-fraud module

cardReader

Status information for XFS4IoT services implementing the CardReader interface. This will be omitted if the CardReader interface is not supported.

cardReader/media

Specifies the media state of the device as one of the following values. This status is independent of any media in the parking stations.

- notSupported - Capability to report media position is not supported by the device (e.g. a typical swipe reader or contactless chip card reader).
- unknown - The media state cannot be determined with the device in its current state (e.g. the value of device is *noDevice*, *powerOff*, *offline* or *hardwareError*).
- present - Media is present in the device, not in the entering position and not jammed. A card in a parking station is not considered to be present. On the latched dip device, this indicates that the card is present in the device and the card is unlatched.
- notPresent - Media is not present in the device and not at the entering position.
- jammed - Media is jammed in the device; operator intervention is required.
- entering - Media is at the entry/exit slot of a motorized device.
- latched - Media is present and latched in a latched dip card unit. This means the card can be used for chip card dialog.

cardReader/retainBin

Specifies the state of the retain bin.

cardReader/security

Specifies the state of the security unit as one of the following:

- notSupported - No security module is available.
- notReady - The security module is not ready to process cards or is inoperable.
- notPresent - The security module is open and ready to process cards.

cardReader/numberCards

The number of cards retained; applicable only to motor driven card units, for non-motorized card units this value is zero. This value is persistent it is reset to zero by the [CardReader.ResetCount](#) command.

cardReader/chipPower

Specifies the state of the chip controlled by this service. Depending on the value of capabilities response, this can either be the chip on the currently inserted user card or the chip on a permanently connected chip card. The state of the chip is one of the following:

- notSupported - Capability to report the state of the chip is not supported by the ID

card unit device. This value is returned for contactless chip card readers.

- unknown - The state of the chip cannot be determined with the device in its current state.
- online - The chip is present, powered on and online (i.e. operational, not busy processing a request and not in an error state).
- busy - The chip is present, powered on, and busy (unable to process an Execute command at this time).
- poweredOff - The chip is present, but powered off (i.e. not contacted).
- noDevice - A card is currently present in the device, but has no chip.
- hardwareError - The chip is present, but inoperable due to a hardware error that prevents it from being used (e.g. MUTE, if there is an unresponsive card in the reader).
- noCard - There is no card in the device.

cardReader/chipModule

Specifies the state of the chip card module reader as one of the following:

- ok - The chip card module is in a good state.
- inoperable - The chip card module is inoperable.
- unknown - The state of the chip card module cannot be determined.
- notSupported - Reporting the chip card module status is not supported.

cardReader/magWriteModule

Specifies the state of the magnetic card writer as one of the following:

- ok - The magnetic card writing module is in a good state.
- inoperable - The magnetic card writing module is inoperable.
- unknown - The state of the magnetic card writing module cannot be determined.
- notSupported - Reporting the magnetic card writing module status is not supported.

cardReader/frontImageModule

Specifies the state of the front image reader as one of the following:

- ok - The front image reading module is in a good state.
- inoperable - The front image reading module is inoperable.
- unknown - The state of the front image reading module cannot be determined.
- notSupported - Reporting the front image reading module status is not supported.

cardReader/backImageModule

Specifies the state of the back image reader as one of the following:

- ok - The back image reading module is in a good state.
- inoperable - The back image reading module is inoperable.
- unknown - The state of the back image reading module cannot be determined.
- notSupported - Reporting the back image reading module status is not supported.

cardReader/parkingStationMedia

An array which contains the states of the parking stations or card stacker module. This is omitted if no parking station and no card stacker module is supported. Each status can be one of the following:

- present - Media is present in the parking station, and not jammed.
- notPresent - Media is not present in the parking station.
- jammed - The parking station is jammed; operator intervention is required.
- notSupported - Reporting the media status in a parking station is not supported by the device.
- unknown - The media state cannot be determined.

cashAcceptor

Status information for XFS4IoT services implementing the CashAcceptor interface. This will be omitted if the CashAcceptor interface is not supported.

cashAcceptor/intermediateStacker

Supplies the state of the intermediate stacker. Following values are possible:

"empty": The intermediate stacker is empty.

"notEmpty": The intermediate stacker is not empty.

"full": The intermediate stacker is full. This may also be reported during a cash-in transaction where a limit specified by CashAcceptor.SetCashInLimit has been reached.

"unknown": Due to a hardware error or other condition, the state of the intermediate stacker cannot be determined.

"notSupported": The physical device has no intermediate stacker.

cashAcceptor/stackerItems

This field informs the client whether items on the intermediate stacker have been in

customer access. Following values are possible:

"customerAccess": Items on the intermediate stacker have been in customer access. If the device is a cash recycler then the items on the intermediate stacker may be there as a result of a previous cash-out operation.

"noCustomerAccess": Items on the intermediate stacker have not been in customer access.

"accessUnknown": It is not known if the items on the intermediate stacker have been in customer access.

"noItems": There are no items on the intermediate stacker or the physical device has no intermediate stacker.

cashAcceptor/banknoteReader

Supplies the state of the banknote reader. Following values are possible:

"ok": The banknote reader is in a good state.

"inoperable": The banknote reader is inoperable.

"unknown": Due to a hardware error or other condition, the state of the banknote reader cannot be determined.

"notSupported": The physical device has no banknote reader.

cashAcceptor/dropBox

The drop box is an area within the CashAcceptor where items which have caused a problem during an operation are stored. This field specifies the status of the drop box. TRUE means that some items are stored in the drop box due to a cash-in transaction which caused a problem. FALSE indicates that the drop box is empty.

cashAcceptor/positions

Array of structures for each position from which items can be accepted.

cashAcceptor/positions/position

Supplies the input or output position as one of the following values:

"inLeft": Left input position.

"inRight": Right input position.

"inCenter": Center input position.

"inTop": Top input position.

"inBottom": Bottom input position.

"inFront": Front input position.

"inRear": Rear input position.

"outLeft": Left output position.

"outRight": Right output position.

"outCenter": Center output position.

"outTop": Top output position.

"outBottom": Bottom output position.

"outFront": Front output position.

"outRear": Rear output position.

cashAcceptor/positions/shutter

Supplies the state of the shutter. Following values are possible:

"closed": The shutter is operational and is closed.

"open": The shutter is operational and is open.

"jammed": The shutter is jammed and is not operational. The field jammedShutterPosition provides the positional state of the shutter.

"unknown": Due to a hardware error or other condition, the state of the shutter cannot be determined.

"notSupported": The physical device has no shutter or shutter state reporting is not supported.

cashAcceptor/positions/positionStatus

The status of the input or output position. Following values are possible:

"empty": The output position is empty.

"notEmpty": The output position is not empty.

"unknown": Due to a hardware error or other condition, the state of the output position

cannot be determined.

"notSupported": The device is not capable of reporting whether or not items are at the output position.

cashAcceptor/positions/transport

Supplies the state of the transport mechanism. The transport is defined as any area leading to or from the position. Following values are possible:

"ok": The transport is in a good state.

"inoperative": The transport is inoperative due to a hardware failure or media jam.

"unknown": Due to a hardware error or other condition the state of the transport cannot be determined.

"notSupported": The physical device has no transport or transport state reporting is not supported.

cashAcceptor/positions/transportStatus

Returns information regarding items which may be on the transport. If the device is a recycler device it is possible that the transport will not be empty due to a previous dispense operation. Following values are possible:

"empty": The transport is empty.

"notEmpty": The transport is not empty.

"notEmptyCustomer": Items which a customer has had access to are on the transport.

"notEmptyUnknown": Due to a hardware error or other condition it is not known whether there are items on the transport.

"notSupported": The device is not capable of reporting whether items are on the transport.

cashAcceptor/positions/jammedShutterPosition

Returns information regarding the position of the jammed shutter. Following values are possible:

"notSupported": The physical device has no shutter or the reporting of the position of a jammed shutter is not supported.

"notJammed": The shutter is not jammed.

"open": The shutter is jammed, but fully open.

"partiallyOpen": The shutter is jammed, but partially open.

"closed": The shutter is jammed, but fully closed.

"unknown": The position of the shutter is unknown.

cashAcceptor/mixedMode

Reports if Mixed Media mode is active. Following values are possible:

"notActive": Mixed Media transactions are not supported by the device or Mixed Media mode is not activated.

"active": Mixed Media mode using the CashAcceptor and ItemProcessor interfaces is activated.

cashDispenser

Status information for XFS4IoT services implementing the CashDispenser interface. This will be omitted if the CashDispenser interface is not supported.

cashDispenser/intermediateStacker

Supplies the state of the intermediate stacker. These bills are typically present on the intermediate stacker as a result of a retract operation or because a dispense has been performed without a subsequent present. Following values are possible:

- empty - The intermediate stacker is empty.
- notEmpty - The intermediate stacker is not empty. The items have not been in customer access.
- notEmptyCustomer - The intermediate stacker is not empty. The items have been in customer access. If the device is a recycler then the items on the intermediate stacker may be there as a result of a previous cash-in operation.
- notEmptyUnknown - The intermediate stacker is not empty. It is not known if the items have been in customer access.
- unknown - Due to a hardware error or other condition, the state of the intermediate stacker cannot be determined.
- notSupported - The physical device has no intermediate stacker.

cashDispenser/positions

Array of structures for each position to which items can be dispensed or presented.

cashDispenser/positions/position

Supplies the output position as one of the following values:

- left - Left output position.
- right - Right output position.
- center - Center output position.
- top - Top output position.
- bottom - Bottom output position.
- front - Front output position.
- rear - Rear output position.

cashDispenser/positions/shutter

Supplies the state of the shutter. Following values are possible:

- closed - The shutter is operational and is closed.
- open - The shutter is operational and is open.
- jammed - The shutter is jammed and is not operational. The field jammedShutterPosition provides the positional state of the shutter.
- unknown - Due to a hardware error or other condition, the state of the shutter cannot be determined.
- notSupported - The physical device has no shutter or shutter state reporting is not supported.

cashDispenser/positions/positionStatus

Returns information regarding items which may be at the output position. If the device is a recycler it is possible that the output position will not be empty due to a previous cash-in operation. Following values are possible:

- empty - The output position is empty.
- notEmpty - The output position is not empty.
- unknown - Due to a hardware error or other condition, the state of the output position cannot be determined.
- notSupported - The device is not capable of reporting whether or not items are at the output position.

cashDispenser/positions/transport

Supplies the state of the transport mechanism. The transport is defined as any area leading to or from the position. Following values are possible:

- ok - The transport is in a good state.

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

- **inoperative** - The transport is inoperative due to a hardware failure or media jam.
- **unknown** - Due to a hardware error or other condition the state of the transport cannot be determined.
- **notSupported** - The physical device has no transport or transport state reporting is not supported.

cashDispenser/positions/transportStatus

Returns information regarding items which may be on the transport. If the device is a recycler device it is possible that the transport will not be empty due to a previous cash-in operation. Following values are possible:

- **empty** - The transport is empty.
- **notEmpty** - The transport is not empty.
- **notEmptyCustomer** - Items which a customer has had access to are on the transport.
- **notEmptyUnknown** - Due to a hardware error or other condition it is not known whether there are items on the transport.
- **notSupported** - The device is not capable of reporting whether items are on the transport.

cashDispenser/positions/jammedShutterPosition

Returns information regarding the position of the jammed shutter. Following values are possible:

- **notSupported** - The physical device has no shutter or the reporting of the position of a jammed shutter is not supported.
- **notJammed** - The shutter is not jammed.
- **open** - The shutter is jammed, but fully open.
- **partiallyOpen** - The shutter is jammed, but partially open.
- **closed** - The shutter is jammed, but fully closed.
- **unknown** - The position of the shutter is unknown.

cashManagement

Status information for XFS4IoT services implementing the CashManagement interface. This will be omitted if the CashManagement interface is not supported.

cashManagement/safeDoor

Supplies the state of the safe door. Following values are possible:

- **doorNotSupported** - Physical device has no safe door or safe door state reporting is not supported.

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

- doorOpen - Safe door is open.
- doorClosed - Safe door is closed.
- doorUnknown - Due to a hardware error or other condition, the state of the safe door cannot be determined.

cashManagement/dispenser

Supplies the state of the logical cash units for dispensing. Following values are possible:

- ok - All cash units present are in a good state.
- cashUnitState - One or more of the cash units is in a low, empty, inoperative or manipulated condition. Items can still be dispensed from at least one of the cash units.
- cashUnitStop - Due to a cash unit failure dispensing is impossible. No items can be dispensed because all of the cash units are in an empty, inoperative or manipulated condition. This state may also occur when a reject/retract cash unit is full or no reject/retract cash unit is present, or when a client lock is set on every cash unit which can be locked.
- cashUnitUnknown - Due to a hardware error or other condition, the state of the cash units cannot be determined.

cashManagement/acceptor

Supplies the state of the cash units for accepting cash. Following values are possible:

- ok - All cash units present are in a good state.
- cashUnitState - One or more of the cash units is in a high, full, inoperative or manipulated condition. Items can still be accepted into at least one of the cash units.
- cashUnitStop - Due to a cash unit failure accepting is impossible. No items can be accepted because all of the cash units are in a full, inoperative or manipulated condition. This state may also occur when a retract cash unit is full or no retract cash unit is present, or when a client lock is set on every cash unit, or when Level 2/3 notes are to be automatically retained within cash units, but all of the designated cash units for storing them are full or inoperative.
- cashUnitUnknown - Due to a hardware error or other condition, the state of the cash units cannot be determined.

keyManagement

Status information for XFS4IoT services implementing the KeyManagement interface. This will be omitted if the KeyManagement interface is not supported.

keyManagement/encryptionState

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Specifies the state of the encryption module.

keyManagement/certificateState

Specifies the state of the public verification or encryption key in the PIN certificate modules.

keyboard

Status information for XFS4IoT services implementing the Keyboard interface. This will be omitted if the Keyboard interface is not supported.

keyboard/autoBeepMode

Specifies whether automatic beep tone on key press is active or not. Active and in-active key beeping is reported independently. autoBeepMode can take a combination of the following values, if the flag is not set auto beeping is not activated (or not supported) for that key type (i.e. active or in-active keys)

textTerminal

Status information for XFS4IoT services implementing the TextTerminal interface. This will be omitted if the TextTerminal interface is not supported.

textTerminal/keyboard

Specifies the state of the keyboard.

textTerminal/keyLock

Specifies the state of the keyboard lock.

textTerminal/displaySizeX

Specifies the horizontal size of the display of the text terminal unit.

textTerminal/displaySizeY

Specifies the vertical size of the display of the text terminal unit.

textTerminal/leds

Specifies array that specifies the state of each LED. Specifies the state of the na, off or a combination of the following flags consisting of one type B, and optionally one type C

textTerminal/leds/na

The Status is not available. Type A

textTerminal/leds/off

The LED is turned off. Type A

textTerminal/leds/slowFlash

The LED is blinking. Type B

textTerminal/leds/mediumFlash

The LED is blinking medium frequency. Type B

textTerminal/leds/quickFlash

The LED is blinking quickly. Type B

textTerminal/leds/continuous

The LED is turned on continuous(steady). Type B

textTerminal/leds/red

The LED is red. Type C

textTerminal/leds/green

The LED is green. Type C

textTerminal/leds/yellow

The LED is yellow. Type C

textTerminal/leds/blue

The LED is blue. Type C

textTerminal/leds/cyan

The LED is cyan. Type C

textTerminal/leds/magenta

The LED is magenta. Type C

textTerminal/leds/white

The LED is white. Type C

printer

Status information for XFS4IoT services implementing the Printer interface. This will be omitted if the Printer interface is not supported.

printer/media

Specifies the state of the print media (i.e. receipt, statement, passbook, etc.) as one of the following values. This field does not apply to journal printers:

- notSupported - The capability to report the state of the print media is not supported by the device.
- unknown - The state of the print media cannot be determined with the device in its current state.
- present - Media is in the print position, on the stacker or on the transport (i.e. a passbook in the parking station is not considered to be present). On devices with continuous paper supplies, this value is set when paper is under the print head. On devices with no supply or individual sheet supplies, this value is set when paper/media is successfully inserted/loaded.
- notPresent - Media is not in the print position or on the stacker.
- jammed - Media is jammed in the device.

- entering - Media is at the entry/exit slot of the device.
- retracted - Media was retracted during the last command which controlled media.

printer/paper

Specifies the state of paper supplies as one of the following values:

- notSupported - Capability not supported by the device.
- unknown - Status cannot be determined with device in its current state.
- full - The paper supply is full.
- low - The paper supply is low.
- out - The paper supply is empty.
- jammed - The paper supply is jammed.

printer/paper/upper

The state of the upper paper supply.

printer/paper/lower

The state of the lower paper supply.

printer/paper/external

The state of the external paper supply.

printer/paper/aux

The state of the auxiliary paper supply.

printer/paper/aux2

The state of the second auxiliary paper supply.

printer/paper/park

The state of the parking station paper supply.

printer/toner

Specifies the state of the toner or ink supply or the state of the ribbon as one of the

following:

- notSupported - Capability not supported by device.
- unknown - Status of toner or ink supply or the ribbon cannot be determined with device in its current state.
- full - The toner or ink supply is full or the ribbon is OK.
- low - The toner or ink supply is low or the print contrast with a ribbon is weak.
- out - The toner or ink supply is empty or the print contrast with a ribbon is not sufficient any more.

printer/ink

Specifies the status of the stamping ink in the printer as one of the following values:

- notSupported - Capability not supported by device.
- unknown - Status of the stamping ink supply cannot be determined with device in its current state.
- full - Ink supply in device is full.
- low - Ink supply in device is low.
- out - Ink supply in device is empty.

printer/lamp

Specifies the status of the printer imaging lamp as one of the following values:

- notSupported - Capability not supported by device.
- unknown - Status of the imaging lamp cannot be determined with device in its current state.
- ok - The lamp is OK.
- fading - The lamp should be changed.
- inop - The lamp is inoperative.

printer/retractBins

An array of bin state objects. If no retain bins are supported, the array will be empty.

printer/retractBins/state

Specifies the state of the printer retract bin as one of the following:

- ok - The retract bin of the printer is in a healthy state.
- full - The retract bin of the printer is full.
- unknown - Status cannot be determined with device in its current state.
- high - The retract bin of the printer is nearly full.

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

- missing - The retract bin is missing.

printer/retractBins/count

The number of media retracted to this bin. This value is persistent; it may be reset to zero by the [Printer.ResetCount](#) command.

printer/mediaOnStacker

The number of media on stacker; applicable only to printers with stacking capability.

printer/paperType

Specifies the type of paper loaded as one of the following:

- unknown - No paper is loaded, reporting of this paper type is not supported or the paper type cannot be determined.
- single - The paper can be printed on only one side.
- dual - The paper can be printed on both sides.

printer/paperType/upper

The upper paper supply paper type.

printer/paperType/lower

The lower paper supply paper type.

printer/paperType/external

The external paper supply paper type.

printer/paperType/aux

The auxililliary paper supply paper type.

printer/paperType/aux2

The second auxililliary paper supply paper type.

printer/paperType/park

The parking station paper supply paper type.

printer/blackMarkMode

Specifies the status of the black mark detection and associated functionality:

- notSupported - Black mark detection is not supported.
- unknown - The status of the black mark detection cannot be determined.
- on - Black mark detection and associated functionality is switched on.
- off - Black mark detection and associated functionality is switched off.

cardEmbosser

Status information for XFS4IoT services implementing the CardEmbosser interface. This will be omitted if the CardEmbosser interface is not supported.

cardEmbosser/media

Specifies the state of the card embosser media as one of the following:

- present - Media is present in the device, not in the entering position and not jammed.
- notPresent - Media is not present in the device and not at the entering position.
- jammed - Media is jammed in the device; operator intervention is required.
- notSupported - Capability to report media position is not supported by the device.
- unknown - The media state cannot be determined with the device in its current state (e.g. the value of `device` is *noDevice*, *powerOff*, *offline*, or *hardwareError*).
- entering - Media is at the entry/exit slot.
- topper - Topper failure.
- inHopper - Card is positioned in input bin.
- outHopper - Card is positioned in output bin.
- msre - Encoding failure.
- retained - Card is positioned in retain bin.

cardEmbosser/retainBin

Specifies the state of the card embosser retain bin as one of the following:

- ok - The retain bin is in a good state.
- full - The retain bin is full.
- high - The retain bin is nearly full.

- notSupported - The retain bin state can not be reported.

cardEmbosser/outputBin

Specifies the state of the card embosser output bin as one of the following:

- ok - The output bin is in a good state.
- full - The output bin is full.
- high - The output bin is nearly full.
- notSupported - The output bin state can not be reported.

cardEmbosser/inputBin

Specifies the state of the card embosser input bin as one of the following:

- ok - The input bin is in a good state.
- empty - The input bin is empty.
- low - The input bin is nearly empty.
- notSupported - The input bin state can not be reported.

cardEmbosser/totalCards

The total number of cards, including those in input bin, output bin, and retain bin.

cardEmbosser/outputCards

The total number of output bin cards.

cardEmbosser/retainCards

The total number of retain bin cards.

cardEmbosser/toner

Specifies the state of the toner or ink supply or the state of the ribbon as one of the following:

- full - The toner or ink supply is full or the ribbon is OK.
- low - The toner or ink supply is low or the print contrast with a ribbon is weak.
- out - The toner or ink supply is empty or the print contrast with a ribbon is not sufficient any more.
- notSupported - The toner or ink supply is not supported by the device.
- unknown - Status of toner or ink supply or the ribbon cannot be determined with

device in its current state.

barcodeReader

Status information for XFS4IoT services implementing the BarcodeReader interface. This will be omitted if the BarcodeReader interface is not supported.

barcodeReader/scanner

Specifies the scanner status (laser, camera or other technology) as one of the following:

- on - Scanner is enabled for reading.
- off - Scanner is disabled.
- inoperative - Scanner is inoperative due to a hardware error.
- unknown - Scanner status cannot be determined.

biometric

Status information for XFS4IoT services implementing the Biometrics interface. This will be omitted if the Biometrics interface is not supported.

biometric/subject

Specifies the state of the subject to be scanned (e.g. finger, palm, retina, etc) as one of the following values:

- present - The subject to be scanned is on the scanning position.
- notPresent - The subject to be scanned is not on the scanning position.
- subjectUnknown - The subject to be scanned cannot be determined with the device in its current state (e.g. the value of *device* is noDevice, powerOff, offline, or hwError).
- subjectNotSupported - The physical device does not support the ability to report whether or not a subject is on the scanning position.

biometric/capture

Indicates whether or not scanned biometric data has been captured using the [Biometric.Read](#) command and is currently stored and ready for comparison. true if data has been captured and is stored, false if no scanned data is present. This will be set to false when scanned data is cleared using the [Biometric.Clear](#) command.

biometric/dataPersistence

Specifies the current data persistence mode. The data persistence mode controls how biometric data that has been captured using the [Biometric.Read](#) command will be handled.

biometric/dataPersistence/persist

Biometric data captured using the [Biometric.Read](#) command can persist until all sessions are closed, the device is power failed or rebooted, or the [Biometric.Read](#) command is requested again. This captured biometric data can also be explicitly cleared using the [Biometric.Clear](#) or [Biometric.Reset](#) commands.

biometric/dataPersistence/clear

Captured biometric data will not persist. Once the data has been either returned in the [Biometric.Read](#) command or used by the [Biometric.Match](#) command, then the data is cleared from the device.

biometric/remainingStorage

Specifies how much of the reserved storage specified by the *templateStorage* capability is remaining for the storage of templates in bytes. This will be zero if not reported.

Event Messages

None

2.2.2 - Common.Capabilities

This command retrieves the capabilities of the device. It may also return vendor specific capability information.

Command Message

| Payload | Type | Required |
|-----------------|---------|----------|
| { | | |
| "timeout": 5000 | integer | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|---|----------------|----------|
| { | | |
| "interfaces": [{}] | array (object) | |
| "name": "Common", | string | |
| "commands": [Add example to YAML], | array (string) | |
| "events": [Add example to YAML], | array (string) | |
| "maximumRequests": 0, | integer | |
| "authenticationRequired": [Add example to YAML] | array (string) | |
| }], | | |
| "common": { | object | |
| "serviceVersion": Add example to YAML, | string | |
| "deviceInformation": [{}] | array (object) | |
| "modelName": Add example to YAML, | string | |
| "serialNumber": Add example to YAML, | string | |
| "revisionNumber": Add example to YAML, | string | |
| "modelDescription": Add example to YAML, | string | |
| "firmware": [{}] | array (object) | |
| "firmwareName": Add example to YAML, | string | |
| "firmwareVersion": Add example to YAML, | string | |
| "hardwareRevision": Add example to YAML | string | |

}],

| | |
|----------------|----------------|
| "software": [{ | array (object) |
|----------------|----------------|

See [firmware](#) properties.

}]

}],

| | |
|----------------------------|--------|
| "vendorModeInformation": { | object |
|----------------------------|--------|

| | |
|-----------------------------|---------|
| "allowOpenSessions": false, | boolean |
|-----------------------------|---------|

| | |
|---|----------------|
| "allowedExecuteCommands": [Add example to YAML] | array (string) |
|---|----------------|

},

| | |
|---------------------------------|----------------|
| "extra": [Add example to YAML], | array (string) |
|---------------------------------|----------------|

| | |
|-------------------|----------------|
| "guideLights": [{ | array (object) |
|-------------------|----------------|

| | |
|----------------|--------|
| "flashRate": { | object |
|----------------|--------|

| | |
|----------------|---------|
| "slow": false, | boolean |
|----------------|---------|

| | |
|------------------|---------|
| "medium": false, | boolean |
|------------------|---------|

| | |
|-----------------|---------|
| "quick": false, | boolean |
|-----------------|---------|

| | |
|---------------------|---------|
| "continuous": false | boolean |
|---------------------|---------|

},

| | |
|------------|--------|
| "color": { | object |
|------------|--------|

| | |
|---------------|---------|
| "red": false, | boolean |
|---------------|---------|

| | |
|-----------------|---------|
| "green": false, | boolean |
|-----------------|---------|

| | |
|------------------|---------|
| "yellow": false, | boolean |
|------------------|---------|

| | |
|----------------|---------|
| "blue": false, | boolean |
|----------------|---------|

| | |
|----------------|---------|
| "cyan": false, | boolean |
|----------------|---------|

| | |
|-------------------|---------|
| "magenta": false, | boolean |
|-------------------|---------|

| | |
|----------------|---------|
| "white": false | boolean |
|----------------|---------|

},

```

"direction": {                                object
  "entry": false,                            boolean
  "exit": false                             boolean
}

}],                                         object

"powerSaveControl": false,                   boolean
"antiFraudModule": false,                   boolean
"synchronizableCommands": [Add example to YAML], array (string)
"endToEndSecurity": false,                  boolean
"hardwareSecurityElement": false,           boolean
"responseSecurityEnabled": false,           boolean
},                                            object

"cardReader": {                                object
  "type": "motor",                           string
  "readTracks": {                                object
    "track1": false,                           boolean
    "track2": false,                           boolean
    "track3": false,                           boolean
    "watermark": false,                        boolean
    "frontTrack1": false,                      boolean
    "frontImage": false,                        boolean
    "backImage": false,                        boolean
    "track1JIS": false,                         boolean
    "track3JIS": false,                         boolean
    "ddi": false,                             boolean
  }
}

```

```

"writeTracks": {                                     object
  "track1": false,                                boolean
  "track2": false,                                boolean
  "track3": false,                                boolean
  "frontTrack1": false,                            boolean
  "track1JIS": false,                             boolean
  "track3JIS": false,                             boolean
},
"chipProtocols": {                                 object
  "chipT0": false,                               boolean
  "chipT1": false,                               boolean
  "chipProtocolNotRequired": false,               boolean
  "chipTypeAPart3": false,                         boolean
  "chipTypeAPart4": false,                         boolean
  "chipTypeB": false,                            boolean
  "chipTypeNFC": false,                           boolean
},
"maxCardCount": 0,                                number
"securityType": "notSupported",                  string
"powerOnOption": "noAction",                     string
"powerOffOption": "noAction",                    string
"fluxSensorProgrammable": false,                 boolean
"readWriteAccessFollowingEject": false,           boolean
"writeMode": {                                    object
  "notSupported": false,                          boolean
  "loco": false,                                boolean
}

```

```

"hico": false,                                boolean
"auto": false,                                 boolean
},
"chipPower": {                                  object
  "notSupported": false,                      boolean
  "cold": false,                             boolean
  "warm": false,                            boolean
  "off": false,                             boolean
},
"memoryChipProtocols": {                      object
  "siemens4442": false,                     boolean
  "gpm896": false,                           boolean
},
"ejectPosition": {                            object
  "exit": false,                            boolean
  "transport": false,                      boolean
},
"numberParkingStations": 0                   integer
},
"cashAcceptor": {                            object
  "type": "tellerBill",                     string
  "maxCashInItems": 0,                      integer
  "shutter": false,                          boolean
  "shutterControl": false,                  boolean
  "intermediateStacker": 0,                 integer
  "itemsTakenSensor": false,                boolean
}

```

```

"itemsInsertedSensor": false,                                boolean
"positions": {                                              object
  "inLeft": false,                                         boolean
  "inRight": false,                                         boolean
  "inCenter": false,                                         boolean
  "inTop": false,                                           boolean
  "inBottom": false,                                         boolean
  "inFront": false,                                         boolean
  "inRear": false,                                          boolean
  "outLeft": false,                                         boolean
  "outRight": false,                                         boolean
  "outCenter": false,                                         boolean
  "outTop": false,                                           boolean
  "outBottom": false,                                         boolean
  "outFront": false,                                         boolean
  "outRear": false,                                          boolean
},
"retractAreas": {                                            object
  "retract": false,                                         boolean
  "transport": false,                                         boolean
  "stacker": false,                                          boolean
  "reject": false,                                           boolean
  "billCassette": false,                                         boolean
  "cashIn": false,                                           boolean
},
"retractTransportActions": {                                 object

```

| | |
|-----------------------------|---------|
| "present": false, | boolean |
| "retract": false, | boolean |
| "reject": false, | boolean |
| "billCassette": false, | boolean |
| "cashIn": false | boolean |
| }, | |
| "retractStackerActions": { | object |
| "present": false, | boolean |
| "retract": false, | boolean |
| "reject": false, | boolean |
| "billCassette": false, | boolean |
| "cashIn": false | boolean |
| }, | |
| "compareSignatures": false, | boolean |
| "replenish": false, | boolean |
| "cashInLimit": { | object |
| "byTotalItems": false, | boolean |
| "byAmount": false, | boolean |
| "multiple": false, | boolean |
| "refuseOther": false | boolean |
| }, | |
| "countActions": { | object |
| "individual": false, | boolean |
| "all": false | boolean |
| }, | |
| "deviceLockControl": false, | boolean |

| | |
|-----------------------------------|---------|
| "mixedMode": false, | boolean |
| "mixedDepositAndRollback": false, | boolean |
| "deplete": false, | boolean |
| "counterfeitAction": "none" | string |
| }, | |
| "cashDispenser": { | object |
| "type": "tellerBill", | string |
| "maxDispenseItems": 0, | integer |
| "shutter": false, | boolean |
| "shutterControl": false, | boolean |
| "retractAreas": { | object |
| "retract": false, | boolean |
| "transport": false, | boolean |
| "stacker": false, | boolean |
| "reject": false, | boolean |
| "itemCassette": false | boolean |
| }, | |
| "retractTransportActions": { | object |
| "present": false, | boolean |
| "retract": false, | boolean |
| "reject": false, | boolean |
| "itemCassette": false | boolean |
| }, | |
| "retractStackerActions": { | object |
| "present": false, | boolean |
| "retract": false, | boolean |

```

"reject": false,                                boolean
"itemCassette": false,                           boolean
},
"intermediateStacker": false,                   boolean
"itemsTakenSensor": false,                      boolean
"positions": {                                  object
  "left": false,                               boolean
  "right": false,                             boolean
  "center": false,                            boolean
  "top": false,                               boolean
  "bottom": false,                            boolean
  "front": false,                            boolean
  "rear": false,                             boolean
},
"moveItems": {                                  object
  "fromCashUnit": false,                      boolean
  "toCashUnit": false,                        boolean
  "toTransport": false,                       boolean
  "toStacker": false,                         boolean
},
"prepareDispense": false,                        boolean
},
"cashManagement": {                            object
  "safeDoor": false,                           boolean
  "cashBox": false,                            boolean
  "exchangeType": {                           object
}

```

| | |
|---|---------|
| " byHand <td>boolean</td> | boolean |
| " toCassettes <td>boolean</td> | boolean |
| " clearRecycler <td>boolean</td> | boolean |
| " depositInto <td>boolean</td> | boolean |
| }, | |
| " itemInfoTypes <td>object</td> | object |
| " serialNumber <td>boolean</td> | boolean |
| " signature <td>boolean</td> | boolean |
| " imageFile <td>boolean</td> | boolean |
| }, | |
| " classificationList <td>boolean</td> | boolean |
| " physicalNoteList <td>boolean</td> | boolean |
| }, | |
| " pinPad <td>object</td> | object |
| " pinFormats <td>object</td> | object |
| " ibm3624 <td>boolean</td> | boolean |
| " ansi <td>boolean</td> | boolean |
| " iso0 <td>boolean</td> | boolean |
| " iso1 <td>boolean</td> | boolean |
| " eci2 <td>boolean</td> | boolean |
| " eci3 <td>boolean</td> | boolean |
| " visa <td>boolean</td> | boolean |
| " diebold <td>boolean</td> | boolean |
| " dieboldCo <td>boolean</td> | boolean |
| " visa3 <td>boolean</td> | boolean |
| " emv <td>boolean</td> | boolean |
| " iso3 <td>boolean</td> | boolean |

| | |
|--|-------------------|
| "ap": false | boolean |
| }, | |
| "presentationAlgorithms": { | object |
| "presentClear": false | boolean |
| }, | |
| "display": { | object |
| "none": false, | boolean |
| "ledThrough": false, | boolean |
| "display": false | boolean |
| }, | |
| "idConnect": false, | boolean |
| "validationAlgorithms": { | object |
| "des": false, | boolean |
| "visa": false | boolean |
| }, | |
| "pinCanPersistAfterUse": false, | boolean |
| "typeCombined": false, | boolean |
| "setPinblockDataRequired": false, | boolean |
| "pinBlockAttributes": [YAML missing items type], | array (undefined) |
| "countrySpecificDK": { | object |
| "protocolSupported": false, | boolean |
| "hsmVendor": Add example to YAML, | string |
| "hsmJournaling": false, | boolean |
| "derivationAlgorithms": { | object |
| "chipZka": false | boolean |
| } | |
| }, | |
| "countrySpecificChinese": { | object |
| "protocolSupported": false | boolean |

```

    },
    "countrySpecificLuxemburg": { object
      "protocolSupported": false boolean
    }
  },
  "crypto": { object
    "algorithms": { object
      "ecb": false, boolean
      "cbc": false, boolean
      "cfb": false, boolean
      "rsa": false, boolean
      "cma": false, boolean
      "desMac": false, boolean
      "triDesEcb": false, boolean
      "triDesCbc": false, boolean
      "triDesCfb": false, boolean
      "triDesMac": false, boolean
      "maaMac": false, boolean
      "triDesMac2805": false, boolean
      "sm4": false, boolean
      "sm4Mac": false boolean
    },
    "emvHashAlgorithm": { object
      "sha1Digest": false, boolean
      "sha256Digest": false boolean
    },
    "cryptoAttributes": [YAML missing items type], array (undefined)
  }
}

```

```

"authenticationAttributes": [YAML missing items type], array (undefined)
"verifyAttributes": [YAML missing items type] array (undefined)
},
"keyManagement": {
  "keyNum": 0, object
  "idKey": { object
    "initialization": false, boolean
    "import": false boolean
  },
  "keyCheckModes": { object
    "self": false, boolean
    "zero": false boolean
  },
  "hsmVendor": Add example to YAML, string
  "rsaAuthenticationScheme": false, boolean
  "2partySig": false, boolean
  "3partyCert": false, boolean
  "3partyCertTr34": false boolean
  "rsaSignatureAlgorithm": { object
    "pkcs1V15": false, boolean
    "pss": false boolean
  },
  "rsaCryptAlgorithm": { object
    "pkcs1V15": false, boolean
    "oaep": false boolean
  },
  "rsaKeyCheckMode": { object
    "sha1": false, boolean
  }
}

```

| | |
|---------------------------------|---------|
| "sha256": false | boolean |
| }, | |
| "signatureScheme": { | object |
| "genRsaKeyPair": false, | boolean |
| "randomNumber": false, | boolean |
| "exportEppId": false, | boolean |
| "enhancedRkl": false | boolean |
| }, | |
| "emvImportSchemes": { | object |
| "plainCA": false, | boolean |
| "chksumCA": false, | boolean |
| "epiCA": false, | boolean |
| "issuer": false, | boolean |
| "icc": false, | boolean |
| "iccPin": false, | boolean |
| "pkcsv15CA": false | boolean |
| }, | |
| "keyBlockImportFormats": { | object |
| "ansTr31KeyBlock": false, | boolean |
| "ansTr31KeyBlockB": false, | boolean |
| "ansTr31KeyBlockC": false | boolean |
| }, | |
| "keyImportThroughParts": false, | boolean |
| "desKeyLength": { | object |
| "single": false, | boolean |
| "double": false, | boolean |
| "triple": false | boolean |

```

    },
    "certificateTypes": { object
        "encKey": false, boolean
        "verificationKey": false, boolean
        "hostKey": false, boolean
    },
    "loadCertOptions": [{ array (object)
        "signer": "certHost", string
        "option": { object
            "newHost": false, boolean
            "replaceHost": false, boolean
        }
    }]
},
"crklLoadOptions": { object
    "noRandom": false, boolean
    "noRandomCrl": false, boolean
    "random": false, boolean
    "randomCrl": false, boolean
},
"restrictedKeyEncKeySupport": [{ array (object)
    "loadingMethod": "rsaAuth2partySig", string
    "uses": { object
        "crypt": false, boolean
        "function": false, boolean
        "macing": false, boolean
        "pinlocal": false, boolean
        "svenckey": false, boolean
    }
}
]
}

```

| | |
|---|-------------------|
| "pinRemote": false | boolean |
| } | |
| }, | |
| "symmetricKeyManagementMethods": { | object |
| "fixedKey": false, | boolean |
| "masterKey": false, | boolean |
| "tdesDukpt": false | boolean |
| }, | |
| "keyAttributes": [YAML missing items type], | array (undefined) |
| "decryptAttributes": [YAML missing items type], | array (undefined) |
| "verifyAttributes": [YAML missing items type] | array (undefined) |
| }, | |
| "keyboard": { | object |
| "autoBeep": { | object |
| "activeAvailable": false, | boolean |
| "activeSelectable": false, | boolean |
| "inactiveAvailable": false, | boolean |
| "inactiveSelectable": false | boolean |
| }, | |
| "etsCaps": [{ | array (object) |
| "xPos": 0, | integer |
| "yPos": 0, | integer |
| "xSize": 0, | integer |
| "ySize": 0, | integer |
| "maximumTouchFrames": 0, | integer |
| "maximumTouchKeys": 0, | integer |
| "floatFlags": { | object |

```

"x": false,                                boolean
"y": false,                                 boolean
}
}]
},
"textTerminal": {                           object
  "type": "fixed",                         string
  "resolutions": [{}                      array (object)
    "sizeX": 0,                            integer
    "sizeY": 0                             integer
  ],
  "keyLock": false,                        boolean
  "displayLight": false,                   boolean
  "cursor": false,                        boolean
  "forms": false,                         boolean
  "charSupport": {                       object
    "ascii": false,                        boolean
    "unicode": false                       boolean
  },
  "leds": [{}                            array (object)
    "off": false,                         boolean
    "slowFlash": false,                   boolean
    "mediumFlash": false,                 boolean
    "quickFlash": false,                  boolean
    "continuous": false,                  boolean
    "red": false,                          boolean
    "green": false,                        boolean
    "yellow": false,                       boolean
  ]
}

```

```

"blue": false,                                boolean
"cyan": false,                                boolean
"magenta": false,                             boolean
"white": false,                               boolean
}
},
"printer": {                                    object
  "type": {                                     object
    "receipt": false,                            boolean
    "passbook": false,                           boolean
    "journal": false,                            boolean
    "document": false,                           boolean
    "scanner": false,                            boolean
  },
  "resolution": {                             object
    "low": false,                               boolean
    "medium": false,                            boolean
    "high": false,                             boolean
    "veryHigh": false,                          boolean
  },
  "readForm": {                                 object
    "ocr": false,                               boolean
    "micr": false,                             boolean
    "msf": false,                               boolean
    "barcode": false,                            boolean
    "pageMark": false,                           boolean
  }
}

```

| | |
|--|---------|
| " readImage <td>boolean</td> | boolean |
| " readEmptyLine <td>boolean</td> | boolean |
| }, | |
| " writeForm <td>object</td> | object |
| " text <td>boolean</td> | boolean |
| " graphics <td>boolean</td> | boolean |
| " ocr <td>boolean</td> | boolean |
| " micr <td>boolean</td> | boolean |
| " msf <td>boolean</td> | boolean |
| " barcode <td>boolean</td> | boolean |
| " stamp <td>boolean</td> | boolean |
| }, | |
| " extents <td>object</td> | object |
| " horizontal <td>boolean</td> | boolean |
| " vertical <td>boolean</td> | boolean |
| }, | |
| " control <td>object</td> | object |
| " eject <td>boolean</td> | boolean |
| " perforate <td>boolean</td> | boolean |
| " cut <td>boolean</td> | boolean |
| " skip <td>boolean</td> | boolean |
| " flush <td>boolean</td> | boolean |
| " retract <td>boolean</td> | boolean |
| " stack <td>boolean</td> | boolean |
| " partialCut <td>boolean</td> | boolean |
| " alarm <td>boolean</td> | boolean |

```

"pageForward": false,                                boolean
"pageBackward": false,                               boolean
"turnMedia": false,                                 boolean
"stamp": false,                                    boolean
"park": false,                                     boolean
"expel": false,                                    boolean
"ejectToTransport": false,                           boolean
"rotate180": false,                                boolean
"clearBuffer": false,                               boolean
},
"maxMediaOnStacker": 0,                             integer
"acceptMedia": false,                               boolean
"multiPage": false,                                boolean
"paperSources": {                                   object
  "upper": false,                                  boolean
  "lower": false,                                  boolean
  "external": false,                               boolean
  "aux": false,                                    boolean
  "aux2": false,                                   boolean
  "park": false,                                    boolean
},
"mediaTaken": false,                                boolean
"retractBins": 0,                                   integer
"maxRetract": [0],                                 array (integer)
"imageType": {                                     object
  "tif": false,                                    boolean
}

```

```

"wmf": false,                                boolean
"bmp": false,                                 boolean
"jpg": false,                                 boolean
},
"frontImageColorFormat": {                     object
"binary": false,                             boolean
"grayscale": false,                         boolean
"full": false,                              boolean
},
"backImageColorFormat": {                     object
"binary": false,                             boolean
"grayScale": false,                          boolean
"full": false,                             boolean
},
"codelineFormat": {                           object
"cmc7": false,                             boolean
"e13b": false,                            boolean
"ocr": false,                             boolean
},
"imageSource": {                            object
"imageFront": false,                        boolean
"imageBack": false,                         boolean
"codeLine": false,                          boolean
},
"dispensePaper": false,                      boolean
"osPrinter": Add example to YAML,           string

```

| | |
|------------------------------------|---------|
| "mediaPresented": false, | boolean |
| "autoRetractPeriod": 0, | integer |
| "retractToTransport": false, | boolean |
| "coercivityType": { | object |
| "low": false, | boolean |
| "high": false, | boolean |
| "auto": false | boolean |
| } | |
| "controlPassbook": { | object |
| "turnForward": false, | boolean |
| "turnBackward": false, | boolean |
| "closeForward": false, | boolean |
| "closeBackward": false | boolean |
| } | |
| "printSides": "notSupp" | string |
| }, | |
| "SensorsAndIndicators": { | object |
| "sensorType": { | object |
| "sensors": false, | boolean |
| "doors": false, | boolean |
| "indicators": false, | boolean |
| "auxiliary": false, | boolean |
| "guidelights": false, | boolean |
| "operatorSwitch": "notAvailable", | string |
| "tamperSensor": "notAvailable", | string |
| "intTamperSensor": "notAvailable", | string |

| | |
|---|-----------------|
| "seismicSensor": "notAvailable", | string |
| "heatSensor": "notAvailable", | string |
| "proximitySensor": "notAvailable", | string |
| "ambientLightSensor": "notAvailable", | string |
| "enhancedAudioSensor": { | object |
| "available": false, | boolean |
| "manual": false, | boolean |
| "auto": false, | boolean |
| "semiAuto": false, | boolean |
| "bidirectional": false | boolean |
| } | |
| "bootSwitchSensor": "notAvailable", | string |
| "displaySensor": "notAvailable", | string |
| "operatorCallButtonSensor": "notAvailable", | string |
| "handsetSensor": { | object |
| "available": false, | boolean |
| "manual": false, | boolean |
| "auto": false, | boolean |
| "semiAuto": false, | boolean |
| "microphone": false | boolean |
| } | |
| "generalInputPort": [false], | array (boolean) |
| "headsetMicrophoneSensor": { | object |
| "available": false, | boolean |
| "manual": false, | boolean |
| "auto": false, | boolean |

```

"semiAuto": false                                boolean
},
"fasciaMicrophoneSensor": "notAvailable",        string
"cabinetDoor": {                                  object
  "available": false,                            boolean
  "closed": false,                             boolean
  "open": false,                               boolean
  "locked": false,                            boolean
  "bolted": false,                            boolean
  "tampered": false,                           boolean
},
"safeDoor": {                                    object
  "available": false,                           boolean
  "closed": false,                            boolean
  "open": false,                               boolean
  "locked": false,                            boolean
  "bolted": false,                            boolean
  "tampered": false,                           boolean
},
"vandalShield": {                                object
  "available": false,                           boolean
  "closed": false,                            boolean
  "open": false,                               boolean
  "locked": false,                            boolean
  "service": false,                            boolean
  "keyboard": false,                           boolean
}

```

```

  "tampered": false           boolean
},
"frontCabinet": {             object
  "available": false,         boolean
  "closed": false,            boolean
  "open": false,              boolean
  "locked": false,            boolean
  "bolted": false,            boolean
  "tampered": false           boolean
},
"rearCabinet": {              object
  "available": false,         boolean
  "closed": false,            boolean
  "open": false,              boolean
  "locked": false,            boolean
  "bolted": false,            boolean
  "tampered": false           boolean
},
"leftCabinet": {              object
  "available": false,         boolean
  "closed": false,            boolean
  "open": false,              boolean
  "locked": false,            boolean
  "bolted": false,            boolean
  "tampered": false           boolean
},

```

| | |
|---|-----------------|
| "rightCabinet": { | object |
| "available": false, | boolean |
| "closed": false, | boolean |
| "open": false, | boolean |
| "locked": false, | boolean |
| "bolted": false, | boolean |
| "tampered": false | boolean |
| }, | |
| "openCloseIndicator": "notAvailable", | string |
| "fasciaLight": "notAvailable", | string |
| "audio": "notAvailable", | string |
| "heating": "notAvailable", | string |
| "consumerDisplayBacklight": "notAvailable", | string |
| "signageDisplay": "notAvailable", | string |
| "transactionIndicator": [false], | array (boolean) |
| "generalOutputPort": [false], | array (boolean) |
| "volume": { | object |
| "available": false, | boolean |
| "volumeLevel": 0 | integer |
| }, | |
| "UPS": { | object |
| "available": false, | boolean |
| "low": false, | boolean |
| "engaged": false, | boolean |
| "powering": false, | boolean |
| "recovered": false | boolean |

```

    },
    "remoteStatusMonitor": "notAvailable",           string
    "audibleAlarm": "notAvailable",                 string
    "enhancedAudioControl": {                      object
        "available": false,                         boolean
        "headsetDetection": false,                  boolean
        "modeControllable": false                  boolean
    },
    "enhancedMicrophoneControlState": {            object
        "available": false,                         boolean
        "headsetDetection": false,                  boolean
        "modeControllable": false                  boolean
    },
    "microphoneVolume": {                          object
        "available": false,                         boolean
        "volumeLevel": 0                           integer
    },
    "autoStartupMode": {                          object
        "available": false,                         boolean
        "specific": false,                         boolean
        "daily": false,                            boolean
        "weekly": false                           boolean
    },
    "guideLights": {                             object
        "cardReader": {                           object
            "flashRate": "notAvailable",          string
        }
    }
}

```

"colour": "red", string
"direction": "entry", string
"position": "default" string
},
"pinPad": { object
See [cardReader](#) properties.
},
"notesDispenser": { object
See [cardReader](#) properties.
},
"coinDispenser": { object
See [cardReader](#) properties.
},
"receiptPrinter": { object
See [cardReader](#) properties.
},
"passbookPrinter": { object
See [cardReader](#) properties.
},
"EnvelopeDepository": { object
See [cardReader](#) properties.
},
"chequeUnit": { object
See [cardReader](#) properties.
},
"billAcceptor": { object

See [cardReader](#) properties.

},

"envelopeDispenser": { object

See [cardReader](#) properties.

},

"documentPrinter": { object

See [cardReader](#) properties.

},

"coinAcceptor": { object

See [cardReader](#) properties.

},

"scanner": { object

See [cardReader](#) properties.

},

"contactless": { object

See [cardReader](#) properties.

},

"cardUnit2": { object

See [cardReader](#) properties.

},

"notesDispenser2": { object

See [cardReader](#) properties.

},

"billAcceptor2": { object

See [cardReader](#) properties.

},

```
"vendorDependent": {  
    See cardReader properties.  
},  
    "flashRate": "notAvailable",  
    string  
    "colour": "red",  
    string  
    "direction": "entry",  
    string  
    "position": "default"  
    string  
}  
}  
},  
    "cardEmbosser": {  
        "compareMagneticStripe": false,  
        boolean  
        "magneticStripeRead": false,  
        boolean  
        "magneticStripeWrite": false,  
        boolean  
        "chipIO": false,  
        boolean  
        "chipProtocol": {  
            "notSupported": false,  
            boolean  
            "chipT0": false,  
            boolean  
            "chipT1": false,  
            boolean  
            "chipProtocolNotRequired": false  
            boolean  
},  
            "charSupport": ,  
            undefined  
            "ascii": false,  
            boolean  
            "unicode": false  
            boolean  
            "type": {  
                "emboss": false,  
                boolean
```

| | |
|----------------------------------|---------|
| "print": false | boolean |
| } | |
| }, | |
| "barcodeReader": { | object |
| "canFilterSymbolologies": false, | boolean |
| "symbolologies": { | object |
| "ean128": false, | boolean |
| "ean8": false, | boolean |
| "ean8_2": false, | boolean |
| "ean8_5": false, | boolean |
| "ean13": false, | boolean |
| "ean13_2": false, | boolean |
| "ean13_5": false, | boolean |
| "jan13": false, | boolean |
| "upcA": false, | boolean |
| "upcE0": false, | boolean |
| "upcE0_2": false, | boolean |
| "upcE0_5": false, | boolean |
| "upcE1": false, | boolean |
| "upcE1_2": false, | boolean |
| "upcE1_5": false, | boolean |
| "upcA_2": false, | boolean |
| "upcA_5": false, | boolean |
| "codabar": false, | boolean |
| "itf": false, | boolean |
| "code11": false, | boolean |

| | |
|---------------------------|---------|
| "code39": false, | boolean |
| "code49": false, | boolean |
| "code93": false, | boolean |
| "code128": false, | boolean |
| "msi": false, | boolean |
| "plessey": false, | boolean |
| "std20f5": false, | boolean |
| "std20f5Iata": false, | boolean |
| "pdf417": false, | boolean |
| "microPdf417": false, | boolean |
| "dataMatrix": false, | boolean |
| "maxiCode": false, | boolean |
| "codeOne": false, | boolean |
| "channelCode": false, | boolean |
| "telepenOriginal": false, | boolean |
| "telepenAim": false, | boolean |
| "rss": false, | boolean |
| "rssExpanded": false, | boolean |
| "rssRestricted": false, | boolean |
| "compositeCodeA": false, | boolean |
| "compositeCodeB": false, | boolean |
| "compositeCodeC": false, | boolean |
| "posiCodeA": false, | boolean |
| "posiCodeB": false, | boolean |
| "triopticCode39": false, | boolean |
| "codablockF": false, | boolean |

```

"code16K": false,                                boolean
"qrCode": false,                                 boolean
"aztec": false,                                 boolean
"ukPost": false,                                boolean
"planet": false,                                boolean
"postnet": false,                               boolean
"canadianPost": false,                           boolean
"netherlandsPost": false,                        boolean
"australianPost": false,                          boolean
"japanesePost": false,                           boolean
"chinesePost": false,                            boolean
"koreanPost": false,                            boolean
}

},
"biometric": {
  "type": {
    "facialFeatures": false,                      boolean
    "voice": false,                                boolean
    "fingerprint": false,                          boolean
    "fingerVein": false,                           boolean
    "iris": false,                                 boolean
    "retina": false,                               boolean
    "handGeometry": false,                         boolean
    "thermalFace": false,                          boolean
    "thermalHand": false,                           boolean
    "palmVein": false,                            boolean
    "signature": false,                           boolean
  }
}

```

```

},  

"compound": false,                                boolean  

"maxCapture": 0,                                  integer  

"templateStorage": Add example to YAML,          string  

"dataFormats": {  

    "isoFid": false,                                boolean  

    "isoFmd": false,                                boolean  

    "ansiFid": false,                                boolean  

    "ansiFmd": false,                                boolean  

    "qso": false,                                    boolean  

    "wso": false,                                    boolean  

    "reservedRaw1": false,                            boolean  

    "reservedTemplate1": false,                      boolean  

    "reservedRaw2": false,                            boolean  

    "reservedTemplate2": false,                      boolean  

    "reservedRaw3": false,                            boolean  

    "reservedTemplate3": false,                      boolean  

},  

"cryptographicalAlgorithm": {  

    "ecb": false,                                    boolean  

    "cbc": false,                                    boolean  

    "cfb": false,                                    boolean  

    "rsa": false,                                    boolean  

},  

"storage": {  

    "secure": false,                                boolean  

    "clear": false,                                 boolean
}

```

```

    },
    "persistenceModes": { object
        "persist": false, boolean
        "clear": false boolean
    },
    "matchSupported": "none", string
    "scanModes": { object
        "scan": false, boolean
        "match": false boolean
    },
    "compareModes": { object
        "verify": false, boolean
        "identity": false boolean
    },
    "clearData": { object
        "scannedData": false, boolean
        "importedData": false, boolean
        "setMatchedData": false boolean
    }
}
}
}

```

Properties

interfaces

Array of interfaces supported by this XFS4IoT service.

interfaces/name

Name of supported XFS4IoT interface.

interfaces/commands

Full array of commands supported by this XFS4IoT interface.

interfaces/events

Full array of events supported by this XFS4IoT interface.

interfaces/maximumRequests

Specifies the maximum number of requests which can be queued by the Service. This will be omitted if not reported. This will be zero if the maximum number of requests is unlimited.

interfaces/authenticationRequired

Array of commands, which need to be authenticated using the security interface.

common

Capability information common to all XFS4IoT services.

common/serviceVersion

Specifies the Service Version.

common/deviceInformation

Array of deviceInformation structures. If the service uses more than one device there will be one array element for each device.

common/deviceInformation/modelName

Specifies the device model name. The property is omitted, if the device model name is unknown.

common/deviceInformation/serialNumber

Specifies the unique serial number of the device. The property is omitted, if the serial number is unknown.

common/deviceInformation/revisionNumber

Specifies the device revision number. The property is omitted, if the device revision number is unknown.

common/deviceInformation/modelDescription

Contains a description of the device. The property is omitted, if the model description is unknown.

common/deviceInformation/firmware

Array of firmware structures specifying the names and version numbers of the firmware that is present. Single or multiple firmware versions can be reported. If the firmware versions are not reported, then this property is omitted.

common/deviceInformation/firmware/firmwareName

Specifies the firmware name. The property is omitted, if the firmware name is unknown.

common/deviceInformation/firmware/firmwareVersion

Specifies the firmware version. The property is omitted, if the firmware version is unknown.

common/deviceInformation/firmware/hardwareRevision

Specifies the hardware revision. The property is omitted, if the hardware revision is unknown.

common/deviceInformation/software

Array of software structures specifying the names and version numbers of the software components that are present. Single or multiple software versions can be reported. If the software versions are not reported, then this property is omitted.

common/deviceInformation/software/firmwareName

Specifies the firmware name. The property is omitted, if the firmware name is unknown.

common/deviceInformation/software/firmwareVersion

Specifies the firmware version. The property is omitted, if the firmware version is unknown.

common/deviceInformation/software/hardwareRevision

Specifies the hardware revision. The property is omitted, if the hardware revision is unknown.

common/vendorModeIformation

Specifies additional information about the Service while in Vendor Dependent Mode. If omitted, all sessions must be closed before entry to VDM.

common/vendorModeIformation/allowOpenSessions

If TRUE, sessions with this Service may remain open during Vendor Dependent Mode for the purposes of monitoring events, sending Info commands, or sending Execute commands listed in lpdwAllowedExecuteCommands. If FALSE, all sessions must be closed before entering Vendor Dependent Mode.

common/vendorModeIformation/allowedExecuteCommands

Array of commands which can be accepted while in Vendor Dependent Mode. Any Execute command which is not included in this list will be rejected with a SequenceError as control of the device has been handed to the Vendor Dependent Client. If omitted, no Execute commands can be accepted.

common/extrा

Specifies a list of vendor-specific, or any other extended, information. The information is returned as a series of "key=value" strings so that it is easily extendable by Service Providers

common/guideLights

Specifies which guidance lights are available

common/guideLights/flashRate

Indicates which flash rates are supported by the guidelight.

common/guideLights/flashRate/slow

The light can blink slowly.

common/guideLights/flashRate/medium

The light can blink medium frequency.

common/guideLights/flashRate/quick

The light can blink quickly.

common/guideLights/flashRate/continuous

The light can be continuous (steady).

common/guideLights/color

Indicates which colors are supported by the guidelight.

common/guideLights/color/red

The light can be red.

common/guideLights/color/green

The light can be green.

common/guideLights/color/yellow

The light can be yellow.

common/guideLights/color/blue

The light can be blue.

common/guideLights/color/cyan

The light can be cyan.

common/guideLights/color/magenta

The light can be magenta.

common/guideLights/color/white

The light can be white.

common/guideLights/direction

Indicates which directions are supported by the guidelight.

common/guideLights/direction/entry

The light can indicate entry.

common/guideLights/direction/exit

The light can indicate exit.

common/powerSaveControl

Specifies whether power saving control is available

common/antiFraudModule

Specifies whether the anti-fraud module is available

common/synchronizableCommands

List of commands that support synchronization.

common/endToEndSecurity

True if this hardware supports End to End security, and requires security tokens as part of the data to secured operations. If true then operations may fail if a valid security token is not supplied.

If false then all operations can be performed without a security token.

common/hardwareSecurityElement

True if this hardware supports End to End security and has a Hardware Security Element which validates the security token. Otherwise false. If this valid is false it may mean that validation is performed in software, or that the device doesn't support End to End security.

common/responseSecurityEnabled

True if this device will return a security token as part of the response data to commands that support End to End security, for example, to validate the result of a dispense operation.

cardReader

Capability information for XFS4IoT services implementing the CardReader interface. This will be omitted if the CardReader interface is not supported.

cardReader/type

Specifies the type of the ID card unit as one of the following:

- motor - The ID card unit is a motor driven card unit.
- swipe - The ID card unit is a swipe (pull-through) card unit.
- dip - The ID card unit is a dip card unit. This dip type is not capable of latching cards entered.
- latchedDip - The ID card unit is a latched dip card unit. This device type is used when a dip card unit device supports chip communication. The latch ensures the consumer cannot remove the card during chip communication. Any card entered will automatically latch when a request to initiate a chip dialog is made (via the [CardReader.ReadRawData](#) command). The [CardReader.EjectCard](#) command is used to unlatch the card.
- contactless - The ID card unit is a contactless card unit, i.e. no insertion of the card is required.

- intelligentContactless - The ID card unit is an intelligent contactless card unit, i.e. no insertion of the card is required and the card unit has built-in EMV or smart card application functionality that adheres to the EMVCo Contactless Specifications or individual payment system's specifications. The ID card unit is capable of performing both magnetic stripe emulation and EMV-like transactions.
- permanent - The ID card unit is dedicated to a permanently housed chip card (no user interaction is available with this type of card).

cardReader/readTracks

Specifies the tracks that can be read by the card reader.

cardReader/readTracks/track1

The card reader can access track 1.

cardReader/readTracks/track2

The card reader can access track 2.

cardReader/readTracks/track3

The card reader can access track 3.

cardReader/readTracks/watermark

The card reader can access the Swedish watermark track.

cardReader/readTracks/frontTrack1

The card reader can access front track 1.

cardReader/readTracks/frontImage

The card reader can read the front image of the card.

cardReader/readTracks/backImage

The card reader can read the back image of the card.

cardReader/readTracks/track1JIS

The card reader can access JIS I track 1.

cardReader/readTracks/track3JIS

The card reader can access JIS I track 3.

cardReader/readTracks/ddi

The card reader can provide dynamic digital identification of the magnetic strip.

cardReader/writeTracks

Specifies the tracks that can be read by the card reader.

cardReader/writeTracks/track1

The card reader can access track 1.

cardReader/writeTracks/track2

The card reader can access track 2.

cardReader/writeTracks/track3

The card reader can access track 3.

cardReader/writeTracks/frontTrack1

The card reader can access front track 1.

cardReader/writeTracks/track1JIS

The card reader can access JIS I track 1.

cardReader/writeTracks/track3JIS

The card reader can access JIS I track 3.

cardReader/chipProtocols

Specifies the chip card protocols that are supported by the card reader.

cardReader/chipProtocols/chipT0

The card reader can handle the T=0 protocol.

cardReader/chipProtocols/chipT1

The card reader can handle the T=0 protocol.

cardReader/chipProtocols/chipProtocolNotRequired

The carder is capable of communicating with the chip without requiring the application to specify any protocol.

cardReader/chipProtocols/chipTypeAPart3

The card reader can handle the ISO 14443 (Part3) Type A contactless chip card protocol.

cardReader/chipProtocols/chipTypeAPart4

The card reader can handle the ISO 14443 (Part4) Type A contactless chip card protocol.

cardReader/chipProtocols/chipTypeB

The card reader can handle the ISO 14443 Type B contactless chip card protocol.

cardReader/chipProtocols/chipTypeNFC

The card reader can handle the ISO 18092 (106/212/424kbps) contactless chip card protocol.

cardReader/maxCardCount

Specifies the maximum numbers of cards that the retain bin and card stacker module bin can hold (zero if not available).

cardReader/securityType

Specifies the type of security module as one of the following:

- notSupported - The device has no security module.
- mm - The security module is a MMBox.
- ``cim86`` - The security module is a CIM86.

cardReader/powerOnOption

Specifies the power-on (or off) capabilities of the device hardware as one of the following options (applicable only to motor driven ID card units):

- noAction - No actions are supported by the device.
- eject - The card will be ejected.
- retain - The card will be retained.
- ejectThenRetain - The card will be ejected for a finite time, then if not taken, the card will be retained. The time for which the card remains ejected is vendor dependent.
- readPosition - The card will be moved to the read position.

cardReader/powerOffOption

Specifies the power-off capabilities of the device hardware. See [powerOnOption](#).

cardReader/fluxSensorProgrammable

Specifies whether the Flux Sensor on the card unit is programmable.

cardReader/readWriteAccessFollowingEject

Specifies whether a card may be read or written after having been pushed to the exit slot with a [CardReader.EjectCard](#) command. The card will be retracted back into the card reader.

cardReader/writeMode

The write capabilities, with respect to whether the device can write low coercivity (loco) and/or high coercivity (hico) magnetic stripes as a combination of the following:

cardReader/writeMode/notSupported

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Does not support writing of magnetic stripes.

cardReader/writeMode/loco

Supports writing of loco magnetic stripes.

cardReader/writeMode/hico

Supports writing of hico magnetic stripes.

cardReader/writeMode/auto

Service Provider is capable of automatically determining whether loco or hico magnetic stripes should be written.

cardReader/chipPower

The chip power management capabilities (in relation to the user or permanent chip controlled by the service, as a combination of the following:

cardReader/chipPower/notSupported

The card reader cannot handle chip power management.

cardReader/chipPower/cold

The card reader can power on the chip and reset it (Cold Reset).

cardReader/chipPower/warm

The card reader can reset the chip (Warm Reset).

cardReader/chipPower/off

The card reader can power off the chip.

cardReader/memoryChipProtocols

The memory card protocols that are supported, as a combination of the following:

cardReader/memoryChipProtocols/siemens4442

The device supports the Siemens 4442 Card Protocol (also supported by the Gemplus GPM2K card).

cardReader/memoryChipProtocols/gpm896

The device supports the Gemplus GPM 896 Card Protocol.

cardReader/ejectPosition

Specifies the target position that is supported for the eject operation, as a combination of the following:

cardReader/ejectPosition/exit

The device can eject a card to the exit position, from which the user can remove it.

cardReader/ejectPosition/transport

The device can eject a card to the transport just behind the exit position, from which the user cannot remove it. The device which supports this must also support the [exit](#) position.

cardReader/numberParkingStations

Specifies the number of supported parking stations or card stackers. If a zero value is specified there is no parking station and no card stacker module supported.

cashAcceptor

Capability information for XFS4IoT services implementing the CashAcceptor interface. This will be omitted if the CashAcceptor interface is not supported.

cashAcceptor/type

Supplies the type of CashAcceptor. Following values are possible:

"tellerBill": The CashAcceptor is a Teller Bill Acceptor.

"selfServiceBill": The CashAcceptor is a Self-Service Bill Acceptor.

"tellerCoin": The CashAcceptor is a Teller Coin Acceptor.

"selfServiceCoin": The CashAcceptor is a Self-Service Coin Acceptor.

cashAcceptor/maxCashInItems

Supplies the maximum number of items that can be accepted in a single CashAcceptor.CashIn command. This value reflects the hardware limitations of the device and therefore it does not change as part of the CashAcceptor.SetCashInLimit command.

cashAcceptor/shutter

If this flag is TRUE then the device has a shutter and explicit shutter control through the commands OpenShutter and CloseShutter is supported. The definition of a shutter will depend on the h/w implementation. On some devices where items are automatically detected and accepted then a shutter is simply a latch that is opened and closed, usually under implicit control by the Service. On other devices, the term shutter refers to a door, which is opened and closed to allow the customer to place the items onto a tray. If a Service cannot detect when items are inserted and there is a shutter on the device, then it must provide explicit client control of the shutter.

cashAcceptor/shutterControl

If set to TRUE the shutter is controlled implicitly by the Service. If set to FALSE the shutter must be controlled explicitly by the client using the OpenShutter and the CloseShutter commands. In either case the PresentMedia command may be used if the presentControl field is reported as FALSE. The shutterControl field is always set to TRUE if the device has no shutter. This field applies to all shutters and all positions.

cashAcceptor/intermediateStacker

Specifies the number of items the intermediate stacker for cash-in can hold. Zero means that there is no intermediate stacker for cash-in available.

cashAcceptor/itemsTakenSensor

Specifies whether or not the CashAcceptor can detect when items at the exit position are taken by the user. If set to TRUE the Service generates an accompanying CashAcceptor.ItemsTaken event. If set to FALSE this event is not generated. This field relates to all output positions.

cashAcceptor/itemsInsertedSensor

Specifies whether the CashAcceptor has the ability to detect when items have actually been inserted by the user. If set to TRUE the Service generates an accompanying CashAcceptor.ItemsInserted event. If set to FALSE this event is not generated. This field relates to all input positions. This flag should not be reported as TRUE unless item insertion can be detected.

cashAcceptor/positions

Specifies the CashAcceptor input and output positions which are available.

cashAcceptor/positions/inLeft

The CashAcceptor has a left input position.

cashAcceptor/positions/inRight

The CashAcceptor has a right input position.

cashAcceptor/positions/inCenter

The CashAcceptor has a center input position.

cashAcceptor/positions/inTop

The CashAcceptor has a top input position.

cashAcceptor/positions/inBottom

The CashAcceptor has a bottom input position.

cashAcceptor/positions/inFront

The CashAcceptor has a front input position.

cashAcceptor/positions/inRear

The CashAcceptor has a rear input position.

`cashAcceptor/positions/outLeft`

The CashAcceptor has a left output position.

`cashAcceptor/positions/outRight`

The CashAcceptor has a right output position.

`cashAcceptor/positions/outCenter`

The CashAcceptor has a center output position.

`cashAcceptor/positions/outTop`

The CashAcceptor has a top output position.

`cashAcceptor/positions/outBottom`

The CashAcceptor has a bottom output position.

`cashAcceptor/positions/outFront`

The CashAcceptor has a front output position.

`cashAcceptor/positions/outRear`

The CashAcceptor has a rear output position.

`cashAcceptor/retractAreas`

Specifies the area to which items may be retracted. If the device does not have a retract capability all flags will be set to false.

`cashAcceptor/retractAreas/retract`

The items may be retracted to a retract cash unit.

cashAcceptor/retractAreas/transport

The items may be retracted to the transport.

cashAcceptor/retractAreas/stacker

The items may be retracted to the intermediate stacker.

cashAcceptor/retractAreas/reject

The items may be retracted to a reject cash unit.

cashAcceptor/retractAreas/billCassette

The items may be retracted to the item cassettes, i.e. cash-in and recycle cash units.

cashAcceptor/retractAreas/cashIn

Items may be retracted to a cash-in cash unit.

cashAcceptor/retractTransportActions

Specifies the actions which may be performed on items which have been retracted to the transport. If the device does not have the capability to retract items to the transport or move items from the transport all flags will be set to false.

cashAcceptor/retractTransportActions/present

The items may be presented.

cashAcceptor/retractTransportActions/retract

The items may be moved to a retract cash unit.

cashAcceptor/retractTransportActions/reject

The items may be moved to a reject bin.

cashAcceptor/retractTransportActions/billCassette

The items may be moved to the item cassettes, i.e. cash-in and recycle cash units.

cashAcceptor/retractTransportActions/cashIn

Items may be retracted to a cash-in cash unit.

cashAcceptor/retractStackerActions

Specifies the actions which may be performed on items which have been retracted to the stacker. If the device does not have the capability to retract items to the stacker or move items from the stacker all flags will be set to false.

cashAcceptor/retractStackerActions/present

The items may be presented.

cashAcceptor/retractStackerActions/retract

The items may be moved to a retract cash unit.

cashAcceptor/retractStackerActions/reject

The items may be moved to a reject bin.

cashAcceptor/retractStackerActions/billCassette

The items may be moved to the item cassettes, i.e. cash-in and recycle cash units.

cashAcceptor/retractStackerActions/cashIn

Items may be retracted to a cash-in cash unit.

cashAcceptor/compareSignatures

Specifies if the Service has the ability to compare signatures through command CashAcceptor.CompareP6Signature.

cashAcceptor/replenish

If set to TRUE the CashAcceptor.ReplenishTarget and CashAcceptor.Replenish commands are supported.

cashAcceptor/cashInLimit

Specifies whether the cash-in limitation is supported or not for the CashAcceptor.SetCashInLimit command. If the device does not have the capability to limit the amount or the number of items during cash-in operations all flags will be set to false.

cashAcceptor/cashInLimit/byTotalItems

The number of successfully processed cash-in items can be limited by specifying the total number of items.

cashAcceptor/cashInLimit/byAmount

The number of successfully processed cash-in items can be limited by specifying the maximum amount of a specific currency.

cashAcceptor/cashInLimit/multiple

CashAcceptor.SetCashInLimit may be called multiple times in a cash-in transaction to update previously specified amount limits. Only valid if combined with "byAmount".

cashAcceptor/cashInLimit/refuseOther

If multiple currencies can be accepted and an amount limit is specified for one or more currencies, any other unspecified currencies are refused. If not specified, there is no amount limit for unspecified currencies. Only valid if specified with "byAmount".

cashAcceptor/countActions

Specifies the count action supported by the CashAcceptor.CashUnitCount command. If the device does not support counting then all flags will be set to false.

cashAcceptor/countActions/individual

The counting of individual cash units via the input structure of the CashAcceptor.CashUnitCount command is supported.

cashAcceptor/countActions/all

The counting of all cash units via the empty payload structure of the CashAcceptor.CashUnitCount command is supported.

cashAcceptor/deviceLockControl

Specifies whether the CashAcceptor supports physical lock/unlock control of the CashAcceptor device and/or the cash units. If this value is set to TRUE, the device and/or the cash units can be locked and unlocked by the CashAcceptor.DeviceLockControl command, and the lock status can be retrieved by the CashAcceptor.DeviceLockStatus command. If this value is set to FALSE, the CashAcceptor will not support the physical lock/unlock control of the CashAcceptor device or the cash units;

cashAcceptor/mixedMode

Specifies whether the device supports accepting and processing items other than the types defined in the CashAcceptor specification. For a description of Mixed Media transactions see section ATM Mixed Media Transaction Flow – Client Guidelines.

cashAcceptor/mixedDepositAndRollback

Specifies whether the device can deposit one type of media and rollback the other in the same Mixed Media transaction. Where mixedDepositAndRollback is TRUE the Service can accept CashAcceptor.CashInEnd and ItemProcessor.MediaInRollback or CashAcceptor.CashInRollback and ItemProcessor.MediaInEnd to complete the current transaction. This value can only be TRUE where mixedMode == TRUE. When mixedDepositAndRollback is FALSE clients must either deposit or return ALL items to complete a transaction. Where Mixed Media transactions are not supported mixedDepositAndRollback is FALSE.

cashAcceptor/deplete

If set to TRUE the CashAcceptor.Deplete command is supported.

cashAcceptor/counterfeitAction

If level 2/3 notes are not to be returned to the customer by these rules, they will not be returned regardless of whether their specific note type is configured to not be accepted by CashAcceptor.ConfigureNotetypes. Following rules are possible:

"none": The device is not able to classify notes as level 1, 2, 3 or 4.

"level2": Notes are classified as level 1, 2, 3 or 4 and only level 2 notes will not be returned to the customer in a cash-in transaction.

"level23": Notes are classified as level 1, 2, 3 or 4 and level 2 and level 3 notes will not be returned to the customer in a cash-in transaction.

cashDispenser

Capability information for XFS4IoT services implementing the CashDispenser interface. This will be omitted if the CashDispenser interface is not supported.

cashDispenser/type

Supplies the type of Dispenser. Following values are possible:

- tellerBill - The Dispenser is a Teller Bill Dispenser.
- selfServiceBill - The Dispenser is a Self-Service Bill Dispenser.
- tellerCoin - The Dispenser is a Teller Coin Dispenser.
- selfServiceCoin - The Dispenser is a Self-Service Coin Dispenser.

cashDispenser/maxDispenseItems

Supplies the maximum number of items that can be dispensed in a single dispense operation. If no limit applies this value will be zero - in this case, if an attempt is made to dispense more items than the hardware limitations will allow, the Service will implement the dispense as a series of sub-dispense operations (see section Sub-Dispensing Command Flow).

cashDispenser/shutter

Specifies whether or not the commands Dispenser.OpenShutter and Dispenser.CloseShutter are supported.

cashDispenser/shutterControl

If set to TRUE the shutter is controlled implicitly by the Service. If set to FALSE the shutter must be controlled explicitly by the client using the Dispenser.OpenShutter and the Dispenser.CloseShutter commands. This field is always set to TRUE if the device has no shutter. This field applies to all shutters and all output positions.

cashDispenser/retractAreas

Specifies the area to which items may be retracted. If the device does not have a retract capability all flags will be set to false.

cashDispenser/retractAreas/retract

The items may be retracted to a retract cash unit.

cashDispenser/retractAreas/transport

The items may be retracted to the transport.

cashDispenser/retractAreas/stacker

The items may be retracted to the intermediate stacker.

cashDispenser/retractAreas/reject

The items may be retracted to a reject cash unit.

cashDispenser/retractAreas/itemCassette

The items may be retracted to the item cassettes, i.e. cassettes that can be dispensed from.

cashDispenser/retractTransportActions

Specifies the actions which may be performed on items which have been retracted to the transport. If the device does not have the capability to retract items to the transport or move items from the transport all flags will be set to false.

cashDispenser/retractTransportActions/present

The items may be presented.

cashDispenser/retractTransportActions/retract

The items may be moved to a retract cash unit.

cashDispenser/retractTransportActions/reject

The items may be moved to a reject bin.

`cashDispenser/retractTransportActions/itemCassette`

The items may be moved to the item cassettes, i.e. cassettes that can be dispensed from.

`cashDispenser/retractStackerActions`

Specifies the actions which may be performed on items which have been retracted to the stacker. If the device does not have the capability to retract items to the stacker or move items from the stacker all flags will be set to false.

`cashDispenser/retractStackerActions/present`

The items may be presented.

`cashDispenser/retractStackerActions/retract`

The items may be moved to a retract cash unit.

`cashDispenser/retractStackerActions/reject`

The items may be moved to a reject bin.

`cashDispenser/retractStackerActions/itemCassette`

The items may be moved to the item cassettes, i.e. cassettes that can be dispensed from.

`cashDispenser/intermediateStacker`

Specifies whether or not the Dispenser supports stacking items to an intermediate position before the items are moved to the exit position. If this value is TRUE, the field "present" of the Dispenser.Dispense command can be set to FALSE.

`cashDispenser/itemsTakenSensor`

Specifies whether the Dispenser can detect when items at the exit position are taken by the user. If set to TRUE the Service generates an accompanying Dispenser.ItemsTakenEvent. If set to FALSE this event is not generated. This field applies to all output positions.

cashDispenser/positions

Specifies the Dispenser output positions which are available.

cashDispenser/positions/left

The Dispenser has a left output position.

cashDispenser/positions/right

The Dispenser has a right output position.

cashDispenser/positions/center

The Dispenser has a center output position.

cashDispenser/positions/top

The Dispenser has a top output position.

cashDispenser/positions/bottom

The Dispenser has a bottom output position.

cashDispenser/positions/front

The Dispenser has a front output position.

cashDispenser/positions/rear

The Dispenser has a rear output position.

cashDispenser/moveItems

Specifies the Dispenser move item options which are available.

cashDispenser/moveItems/fromCashUnit

The Dispenser can dispense items from the cash units to the intermediate stacker while there are items on the transport.

cashDispenser/moveItems/toCashUnit

The Dispenser can retract items to the cash units while there are items on the intermediate stacker.

cashDispenser/moveItems/toTransport

The Dispenser can retract items to the transport while there are items on the intermediate stacker.

cashDispenser/moveItems/toStacker

The Dispenser can dispense items from the cash units to the intermediate stacker while there are already items on the intermediate stacker that have not been in customer access. Items remaining on the stacker from a previous dispense may first need to be rejected explicitly by the client if they are not to be presented.

cashDispenser/prepareDispense

On some hardware it can take a significant amount of time for the dispenser to get ready to dispense media. On this type of hardware the Dispenser.PrepareDispense command can be used to improve transaction performance. This flag indicates if the hardware requires the client to use the Dispenser.PrepareDispense command to maximize transaction performance. If this flag is TRUE then the Dispenser.PrepareDispense command is supported and can be used to improve transaction performance. If this flag is FALSE then the Dispenser.PrepareDispense command is not supported.

cashManagement

Capability information for XFS4IoT services implementing the CashManagement interface. This will be omitted if the CashManagement interface is not supported.

cashManagement/safeDoor

Specifies whether or not the CashManagement.OpenSafeDoor command is supported.

cashManagement/cashBox

This field is only applicable to teller type devices. It specifies whether or not tellers have been assigned a cash box.

cashManagement/exchangeType

Specifies the type of cash unit exchange operations supported by the device.

cashManagement/exchangeType/byHand

The device supports manual replenishment either by filling the cash unit by hand or by replacing the cash unit.

cashManagement/exchangeType/toCassettes

The device supports moving items from the replenishment cash unit to another cash unit.

cashManagement/exchangeType/clearRecycler

The device supports the emptying of recycle cash units.

cashManagement/exchangeType/depositInto

The device supports moving items from the deposit entrance to the bill cash units.

cashManagement/itemInfoTypes

Specifies the types of information that can be retrieved through the CashManagement.GetItemInfo command.

cashManagement/itemInfoTypes/serialNumber

Serial Number of the item.

cashManagement/itemInfoTypes/signature

Signature of the item.

cashManagement/itemInfoTypes/imageFile

Image file of the item.

cashManagement/classificationList

Specifies whether the device has the capability to maintain a classification list of serial numbers as well as supporting the associated operations. This can either be TRUE if the device has the capability or FALSE if it does not.

cashManagement/physicalNoteList

Specifies whether the Service supports note number lists on physical cash units. This can either be TRUE if the Service has the capability or FALSE if it does not.

pinPad

Capability information for XFS4IoT services implementing the PinPad interface. This will be omitted if the PinPad interface is not supported.

pinPad/pinFormats

Supported PIN format.

pinPad/pinFormats/ibm3624

PIN left justified, filled with padding characters, PIN length 4-16 digits. The padding character is a hexadecimal digit in the range 0x00 to 0x0F.

pinPad/pinFormats/ansi

PIN is preceded by 0x00 and the length of the PIN (0x04 to 0x0C), filled with padding character 0x0F to the right, PIN length 4-12 digits, XORed with PAN (Primary Account Number, minimum 12 digits without check number).

pinPad/pinFormats/iso0

PIN is preceded by 0x00 and the length of the PIN (0x04 to 0x0C), filled with padding character 0x0F to the right, PIN length 4-12 digits, XORed with PAN (Primary Account Number without check number, no minimum length specified, missing digits are filled with 0x00).

pinPad/pinFormats/iso1

PIN is preceded by 0x01 and the length of the PIN (0x04 to 0x0C), padding characters are taken from a transaction field (10 digits).

pinPad/pinFormats/eci2

PIN left justified, filled with padding characters, PIN only 4 digits.

pinPad/pinFormats/eci3

PIN is preceded by the length (digit), PIN length 4-6 digits, the padding character can range from 0x0 through 0xF".

pinPad/pinFormats/visa

PIN is preceded by the length (digit), PIN length 4-6 digits. If the PIN length is less than six digits the PIN is filled with 0x0 to the length of six, the padding character can range from 0x0 through 0x9 (This format is also referred to as VISA2).

pinPad/pinFormats/diebold

PIN is padded with the padding character and may be not encrypted, single encrypted or double encrypted.

pinPad/pinFormats/dieboldCo

PIN with the length of 4 to 12 digits, each one with a value of 0x0 to 0x9, is preceded by the one-digit coordination number with a value from 0x0 to 0xF, padded with the padding character with a value from 0x0 to 0xF and may be not encrypted, single encrypted or double encrypted.

pinPad/pinFormats/visa3

PIN with the length of 4 to 12 digits, each one with a value of 0x0 to 0x9, is followed by a delimiter with the value of 0xF and then padded by the padding character with a value between 0x0 to 0xF.

pinPad/pinFormats/env

The PIN block is constructed as follows: PIN is preceded by 0x02 and the length of the PIN

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

(0x04 to 0x0C), filled with padding character 0x0F to the right, formatted up to 248 bytes of other data as defined within the EMV 4.0 specifications and finally encrypted with an RSA key.

pinPad/pinFormats/iso3

PIN is preceded by 0x03 and the length of the PIN (0x04 to 0x0C), padding characters sequentially or randomly chosen, XORed with digits from PAN.

pinPad/pinFormats/ap

PIN is formatted according to the Italian Bancomat specifications. It is known as the Authentication Parameter PIN block and is created with a 5 digit PIN, an 18 digit PAN, and the 8 digit CCS from the track data.

pinPad/presentationAlgorithms

Supported presentation algorithms.

pinPad/presentationAlgorithms/presentClear

Algorithm for the presentation of a clear text PIN to a chipcard. Each digit of the clear text PIN is inserted as one nibble (=halfbyte) into ChipData.

pinPad/display

Specifies the type of the display used in the PIN pad module.

pinPad/display/none

No display unit.

pinPad/display/ledThrough

Lights next to text guide user.

pinPad/display/display

A real display is available (this doesn't apply for self-service).

pinPad/idConnect

Specifies whether the PIN pad is directly physically connected to the ID card unit. If the value is true, the PIN will be transported securely during the command [Pinpad.PresentIdc](#).

pinPad/validationAlgorithms

Specifies the algorithms for PIN validation supported by the service.

pinPad/validationAlgorithms/des

DES algorithm.

pinPad/validationAlgorithms/visa

Visa algorithm.

pinPad/pinCanPersistAfterUse

Specifies whether the device can retain the PIN after a PIN processing command.

pinPad/typeCombined

Specifies whether the keypad used in the secure PIN pad module is integrated within a generic Win32 keyboard. true means the secure PIN keypad is integrated within a generic Win32 keyboard and standard Win32 key events will be generated for any key when there is no active [Keyboard.GetData](#) or [Keyboard.GetPin](#) command. Note that XFS continues to support defined PIN keys only, and is not extended to support new alphanumeric keys.

pinPad/setPinblockDataRequired

Specifies whether the command [Pinpad.SetPinblockData](#) must be called before the PIN is entered via [Keyboard.GetPin](#) and retrieved via [Pinpad.GetPinblock](#).

pinPad/pinBlockAttributes

Array of attributes supported by the [Pinpad.GetPinblock](#) command.

pinPad/countrySpecificDK

Specified capabilities of German specific protocol supports.

pinPad/countrySpecificDK/protocolSupported

Specifies whether the device supports the DK (Deutsche Kreditwirtschaft) formerly known as the ZKA (Zentraler Kreditausschuss) protocol or not.

pinPad/countrySpecificDK/hsmVendor

Identifies the hsm Vendor. hsmVendor is an empty string or this field is not set when the hsm Vendor is unknown or the HSM is not supported.

pinPad/countrySpecificDK/hsmJournaling

Specifies whether the hsm supports journaling by the [Pinpad.DK.GetJournal](#) command. The value of this parameter is either TRUE or FALSE. TRUE means the hsm supports journaling by [Pinpad.DK.GetJournal](#).

pinPad/countrySpecificDK/derivationAlgorithms

Supported derivation algorithms.

pinPad/countrySpecificDK/derivationAlgorithms/chipZka

Algorithm for the derivation of a chip card individual key as described by the German ZKA.

pinPad/countrySpecificChinese

Specified capabilities of Chinese specific PBOC3.0 protocol supports.

pinPad/countrySpecificChinese/protocolSupported

Specifies whether device supports the protocol for China commands or not. The reference for this specific protocol are the Financial industry standard of the People's Republic of China PBOC3.0 JR/T 0025 and the Password industry standard of the People's Republic of China GM/T 0003, GM/T 004.

pinPad/countrySpecificLuxemburg

Specified capabilities of Luxemburg specific protocol supports.

pinPad/countrySpecificLuxemburg/protocolSupported

Specifies whether the device supports Protocol for Luxemburg commands or not. The reference for this specific protocol is the Authorization Center in Luxemburg.

crypto

Capability information for XFS4IoT services implementing the Crypto interface. This will be omitted if the Crypto interface is not supported.

crypto/algorithms

Supported encryption modes.

crypto/algorithms/ecb

Electronic Code Book.

crypto/algorithms/cbc

Cipher Block Chaining.

crypto/algorithms/cfb

Cipher Feed Back.

crypto/algorithms/rsa

RSA Encryption.

crypto/algorithms/cma

ECMA Encryption.

crypto/algorithms/desMac

MAC calculation using CBC.

crypto/algorithms/triDesEcb

Triple DES with Electronic Code Book.

crypto/algorithms/triDesCbc

Triple DES with Cipher Block Chaining.

crypto/algorithms/triDesCfb

Triple DES with Cipher Feed Back.

crypto/algorithms/triDesMac

Last Block Triple DES MAC as defined in ISO/IEC 9797-1:1999 [Ref. 32], using: block length n=64, padding Method 1 (when padding=0), MAC Algorithm 3, MAC length m where 32<=m<=64.

crypto/algorithms/maaMac

MAC calculation using the Message authenticator algorithm as defined in ISO 8731-2.

crypto/algorithms/triDesMac2805

Triple DES MAC calculation as defined in ISO 16609:2004 and and Australian Standard 2805.4.

crypto/algorithms/sm4

SM4 block cipher algorithm as defined in Password industry standard of the People's Republic of China GM/T 0002-2012.

crypto/algorithms/sm4Mac

EMAC calculation using the Message authenticator algorithm as defined in as defined in Password industry standard of the People's Republic of China GM/T 0002-2012. and and in PBOC3.0 JR/T 0025.17-2013.

crypto/envHashAlgorithm

Specifies which hash algorithm is supported for the calculation of the HASH.

crypto/envHashAlgorithm/sha1Digest

The SHA 1 digest algorithm is supported by the [Crypto.Digest](#) command.

crypto/envHashAlgorithm/sha256Digest

The SHA 256 digest algorithm, as defined in ISO/IEC 10118-3:2004 and FIPS 180-2, is supported by the [Crypto.Digest](#) command.

crypto/cryptoAttributes

Array of attributes supported by the [Crypto.CryptoData](#) command.

crypto/authenticationAttributes

Array of attributes supported by the [Crypto.GenerateAuthentication](#) command.

crypto/verifyAttributes

Array of attributes supported by the [Crypto.VerifyAuthentication](#) command.

keyManagement

Capability information for XFS4IoT services implementing the KeyManagement interface. This will be omitted if the KeyManagement interface is not supported.

keyManagement/keyNum

Number of the keys which can be stored in the encryption/decryption module.

keyManagement/idKey

Specifies if key owner identification (in commands referenced as lpxIdent), which authorizes access to the encryption module, is required. A zero value is returned if the

encryption module does not support this capability.

keyManagement/idKey/initialization

ID key is returned by the [KeyManagement.Initialization](#) command.

keyManagement/idKey/import

ID key is required as input for the [KeyManagement.ImportKey](#) and [KeyManagement.DeriveKey](#) command.

keyManagement/keyCheckModes

Specifies the key check modes that are supported to check the correctness of an imported key value.

keyManagement/keyCheckModes/self

The key check value is created by an encryption of the key with itself. For a double-length or triple-length key the kcv is generated using 3DES encryption using the first 8 bytes of the key as the source data for the encryption.

keyManagement/keyCheckModes/zero

The key check value is created by encrypting a zero value with the key.

keyManagement/hsmVendor

Identifies the hsm Vendor. hsmVendor is an empty string or this property is not set when the hsm Vendor is unknown or the HSM is not supported.

keyManagement/rsaAuthenticationScheme

Specifies which type of Remote Key Loading/Authentication.

keyManagement/rsaAuthenticationScheme/2partySig

Two-party Signature based authentication.

keyManagement/rsaAuthenticationScheme/3partyCert

Three-party Certificate based authentication.

keyManagement/rsaAuthenticationScheme/3partyCertTr34

Three-party Certificate based authentication described by X9 TR34-2012.

keyManagement/rsaSignatureAlgorithm

Specifies which type of rsa Signature Algorithm.

keyManagement/rsaSignatureAlgorithm/pkcs1V15

pkcs1V15 Signatures supported.

keyManagement/rsaSignatureAlgorithm/pss

pss Signatures supported.

keyManagement/rsaCryptAlgorithm

Specifies which type of rsa Encipherment Algorithm.

keyManagement/rsaCryptAlgorithm/pkcs1V15

pkcs1V15 algorithm supported.

keyManagement/rsaCryptAlgorithm/oaep

oaep algorithm supported.

keyManagement/rsaKeyCheckMode

Specifies which algorithm/method used to generate the public key check value/thumb print.

keyManagement/rsaKeyCheckMode/sha1

sha1 is supported as defined in Ref. 3.

keyManagement/rsaKeyCheckMode/sha256

sha256 is supported as defined in ISO/IEC 10118-3:2004 and FIPS 180-2.

keyManagement/signatureScheme

Specifies which capabilities are supported by the Signature scheme.

keyManagement/signatureScheme/genRsaKeyPair

Specifies if the Service Provider supports the rsa Signature Scheme

[KeyManagement.GenerateRSAKeyPair](#) and [KeyManagement.ExportRSAEPPSignedItem](#) commands.

keyManagement/signatureScheme/randomNumber

Specifies if the Service Provider returns a random number from the StartKeyExchange GE command within the rsa Signature Scheme.

keyManagement/signatureScheme/exportEppId

Specifies if the Service Provider supports exporting the EPP Security Item within the rsa Signature Scheme.

keyManagement/signatureScheme/enhancedRkl

Specifies that the Service Provider supports the Enhanced Signature Remote Key Scheme. This scheme allows the customer to manage their own public keys independently of the Signature Issuer. When this mode is supported then the key loaded signed with the Signature Issuer key is the host root public key PKROOT, rather than PKHOST.

keyManagement/emuImportSchemes

Identifies the supported emu Import Scheme(s).

keyManagement/emuImportSchemes/plainCA

A plain text CA public key is imported with no verification.

keyManagement/emvImportSchemes/chksumCA

A plain text CA public key is imported using the EMV 2000 verification algorithm.

keyManagement/emvImportSchemes/epiCA

A CA public key is imported using the selfsign scheme defined in the Europay International, epi CA Module Technical - Interface specification."

keyManagement/emvImportSchemes/issuer

An Issuer public key is imported as defined in EMV 2000 Book II.

keyManagement/emvImportSchemes/icc

An ICC public key is imported as defined in EMV 2000 Book II.

keyManagement/emvImportSchemes/iccPin

An ICC PIN public key is imported as defined in EMV 2000 Book II.

keyManagement/emvImportSchemes/pkcsv15CA

A CA public key is imported and verified using a signature generated with a private key for which the public key is already loaded..

keyManagement/keyBlockImportFormats

Supported key block formats.

keyManagement/keyBlockImportFormats/ansTr31KeyBlock

Supports ANS TR-31A Keyblock format key import.

keyManagement/keyBlockImportFormats/ansTr31KeyBlockB

Supports ANS TR-31B Keyblock format key import.

keyManagement/keyBlockImportFormats/ansTr31KeyBlockC

Supports ANS TR-31C Keyblock format key import.

keyManagement/keyImportThroughParts

Specifies whether the device is capable of importing keys in multiple parts. TRUE means the device supports the key import in multiple parts.

keyManagement/desKeyLength

Specifies which length of DES keys are supported.

keyManagement/desKeyLength/single

8 byte DES keys are supported.

keyManagement/desKeyLength/double

16 byte DES keys are supported.

keyManagement/desKeyLength/triple

24 byte DES keys are supported.

keyManagement/certificateTypes

Specifies supported certificate types.

keyManagement/certificateTypes/encKey

Supports the EPP public encryption certificate.

keyManagement/certificateTypes/verificationKey

Supports the EPP public verification certificate.

keyManagement/certificateTypes/hostKey

Supports the Host public certificate.

keyManagement/loadCertOptions

Specifying the options supported by the [KeyManagement.LoadCertificate](#) command.

keyManagement/loadCertOptions/signer

Specifies the signers supported by the [KeyManagement.LoadCertificate](#) command. The possible variables are:

- certHost - The current Host RSA Private Key is used to sign the token.
- sigHost - The current Host RSA Private Key is used to sign the token, signature format is used.
- hl - A Higher-Level Authority RSA Private Key is used to sign the token.
- certHostTr34 - The current Host RSA Private Key is used to sign the token, compliant with X9 TR34-2012.
- caTr34 - The Certificate Authority RSA Private Key is used to sign the token, compliant with X9 TR34-2012.
- hlTr34 - A Higher-Level Authority RSA Private Key is used to sign the token, compliant with X9 TR34-2012."

keyManagement/loadCertOptions/option

Specifies the load options supported by the [KeyManagement.LoadCertificate](#) command.

keyManagement/loadCertOptions/option/newHost

Load a new Host certificate, where one has not already been loaded.

keyManagement/loadCertOptions/option/replaceHost

Replace the epp to a new Host certificate, where the new Host certificate is signed by signer.

keyManagement/crklLoadOptions

Supported options to load the Key Transport Key using the Certificate Remote Key Loading protocol.

keyManagement/crklLoadOptions/noRandom

Import a Key Transport Key without generating and using a random number.

keyManagement/crklLoadOptions/noRandomCrl

Import a Key Transport Key with a Certificate Revocation List appended to the input message. A random number is not generated nor used.

keyManagement/crklLoadOptions/random

Import a Key Transport Key by generating and using a random number.

keyManagement/crklLoadOptions/randomCrl

Import a Key Transport Key with a Certificate Revocation List appended to the input parameter. A random number is generated and used.

keyManagement/restrictedKeyEncKeySupport

A array of object specifying the loading methods that support the RestrictedKeyEncKey usage flag and the allowable usage flag combinations.

keyManagement/restrictedKeyEncKeySupport/loadingMethod

Specifies the loading methods supported. The possible variables are:

- rsaAuth2partySig - Two-party Signature based.
- rsaAuth3partyCert - Three-party Certificate based.
- rsaAuth3partyCertTr34 - Three- party Certificate based TR34.
- restrictedSecurekeyentry - Restricted secure key entry.

keyManagement/restrictedKeyEncKeySupport/uses

Specifies one or more usage flags that can be used in combination with the RestrictedKeyEncKey.

keyManagement/restrictedKeyEncKeySupport/uses/crypt

Key is used for encryption and decryption.

keyManagement/restrictedKeyEncKeySupport/uses/function

Key is used for Pin block creation.

keyManagement/restrictedKeyEncKeySupport/uses/macing

Key is using for macing.

keyManagement/restrictedKeyEncKeySupport/uses/pinlocal

Key is used only for local PIN check.

keyManagement/restrictedKeyEncKeySupport/uses/svenckey

Key is used as cbc start Value encryption key.

keyManagement/restrictedKeyEncKeySupport/uses/pinRemote

Key is used only for PIN block creation.

keyManagement/symmetricKeyManagementMethods

Specifies the Symmentric Key Management modes.

keyManagement/symmetricKeyManagementMethods/fixedKey

This method of key management uses fixed keys for transaction processing.

keyManagement/symmetricKeyManagementMethods/masterKey

This method uses a hierarchy of Key Encrypting Keys and Transaction Keys. The highest level of Key Encrypting Key is known as a Master Key. Transaction Keys are distributed and replaced encrypted under a Key Encrypting Key.

keyManagement/symmetricKeyManagementMethods/tdesDukpt

This method uses TDES Derived Unique Key Per Transaction (see reference 45).

keyManagement/keyAttributes

Array of attributes supported by [KeyManagement.ImportKey](#) command for the key to be loaded.

keyManagement/decryptAttributes

Array of attributes supported by the Import command for the key used to decrypt or unwrap the key being imported.

keyManagement/verifyAttributes

Array of attributes supported by Import command for the key used for verification before importing the key."

keyboard

Capability information for XFS4IoT services implementing the Keyboard interface. This will be omitted if the Keyboard interface is not supported.

keyboard/autoBeep

Specifies whether the device will emit a key beep tone on key presses of active keys or inactive keys, and if so, which mode it supports

keyboard/autoBeep/activeAvailable

Automatic beep tone on active key key-press is supported. If this flag is not set then automatic beeping for active keys is not supported.

keyboard/autoBeep/activeSelectable

Automatic beeping for active keys can be controlled turned on and off by the client. If this flag is not set then automatic beeping for active keys cannot be controlled by a client.

keyboard/autoBeep/inactiveAvailable

Automatic beep tone on in-active key keypress is supported. If this flag is not set then automatic beeping for in-active keys is not supported.

keyboard/autoBeep/inactiveSelectable

Automatic beeping for in-active keys can be controlled turned on and off by the client. If this flag is not set then automatic beeping for in-active keys cannot be controlled by a client.

keyboard/etsCaps

Specifies the capabilities of the ets device.

keyboard/etsCaps/xPos

Specifies the position of the left edge of the ets in Windows virtual screen coordinates. This value may be negative because the of the monitor position on the virtual desktop.

keyboard/etsCaps/yPos

Specifies the position of the right edge of the ets in Windows virtual screen coordinates. This value may be negative because the of the monitor position on the virtual desktop.

keyboard/etsCaps/xSize

Specifies the width of the ets in Windows virtual screen coordinates.

keyboard/etsCaps/ySize

Specifies the height of the ets in Windows virtual screen coordinates.

keyboard/etsCaps/maximumTouchFrames

Specifies the maximum number of Touch-Frames that the device can support in a touch keyboard definition.

keyboard/etsCaps/maximumTouchKeys

Specifies the maximum number of Touch-Keys that the device can support within any a touchframe.

keyboard/etsCaps/floatFlags

Specifies if the device can float the touch keyboards. FloatNone if the PIN device cannot randomly shift the layout.

keyboard/etsCaps/floatFlags/x

Specifies that the PIN device will randomly shift the layout in a horizontal direction

keyboard/etsCaps/floatFlags/y

Specifies that the PIN device will randomly shift the layout in a vertical direction.

textTerminal

Capability information for XFS4IoT services implementing the TextTerminal interface. This will be omitted if the TextTerminal interface is not supported.

textTerminal/type

Specifies the type of the text terminal unit.

textTerminal/resolutions

Array specifies the resolutions supported by the physical display device. (For the definition of Resolution see the command [TextTerminal.SetResolution](#)). The resolution indicated in the first position is the default resolution and the device will be placed in this resolution when the Service Provider is initialized or reset through the [TextTerminal.Reset](#) command.

textTerminal/resolutions/sizeX

TSpecifies the horizontal size of the display of the text terminal unit (the number of columns that can be displayed).

textTerminal/resolutions/sizeY

Specifies the vertical size of the display of the text terminal unit (the number of rows that can be displayed).

textTerminal/keyLock

Specifies whether the text terminal unit has a key lock switch.

textTerminal/displayLight

Specifies whether the text terminal unit has a display light that can be switched ON and OFF with the [TextTerminal.DispLight](#) command.

textTerminal/cursor

Specifies whether the text terminal unit display supports a cursor.

textTerminal/forms

Specifies whether the text terminal unit service supports forms oriented input and output.

textTerminal/charSupport

For charSupport, a Service Provider can support ONLY ascii forms or can support BOTH ascii and unicode forms. A Service Provider can not support UNICODE forms without also supporting ASCII forms."

textTerminal/charSupport/ascii

Ascii is supported for forms.

textTerminal/charSupport/unicode

Unicode is supported for forms.

textTerminal/leds

Specifies which LEDs are available. The elements of this array are specified as a combination of the following flags and indicate all of the possible flash rates (type B) and colors (type C) that the LED is capable of handling. If the LED only supports one color then no value of type C is returned.

textTerminal/leds/off

The LED can be off. Type:(A)

textTerminal/leds/slowFlash

The LED can be blinking. Type:(B)

textTerminal/leds/mediumFlash

The LED can be blinking medium frequency. Type:(B)

textTerminal/leds/quickFlash

The LED can be blinking quickly. Type:(B)

textTerminal/leds/continuous

The LED can be turned on continuous(steady). Type:(B)

textTerminal/leds/red

The LED can be red. Type:(C)

textTerminal/leds/green

The LED can be green. Type:(C)

textTerminal/leds/yellow

The LED can be yellow. Type:(C)

textTerminal/leds/blue

The LED can be blue. Type:(C)

textTerminal/leds/cyan

The LED can be cyan. Type:(C)

textTerminal/leds/magenta

The LED can be magenta. Type:(C)

textTerminal/leds/white

The LED can be white. Type:(C)

printer

Capability information for XFS4IoT services implementing the Printer interface. This will be omitted if the Printer interface is not supported.

printer/type

Specifies the type(s) of the physical device driven by the logical service.

printer/type/receipt

The device is a receipt printer.

printer/type/passbook

The device is a passbook printer.

printer/type/journal

The device is a journal printer.

printer/type/document

The device is a document printer.

printer/type/scanner

The device is a scanner that may have printing capabilities.

printer/resolution

Specifies at which resolution(s) the physical device can print. Used by the client to select the level of print quality desired; does not imply any absolute level of resolution, only

relative.

printer/resolution/low

The device can print low resolution.

printer/resolution/medium

The device can print medium resolution.

printer/resolution/high

The device can print high resolution.

printer/resolution/veryHigh

The device can print very high resolution.

printer/readForm

Specifies whether the device can read data from media, as a combination of the following flags.

printer/readForm/ocr

Device has OCR capability.

printer/readForm/micr

Device has MICR capability.

printer/readForm/msf

Device has MSF capability.

printer/readForm/barcode

Device has Barcode capability.

printer/readForm/pageMark

Device has Page Mark capability.

printer/readForm/readImage

Device has imaging capability.

printer/readForm/readEmptyLine

Device has capability to detect empty print lines for passbook printing.

printer/writeForm

Specifies whether the device can write data to the media, as a combination of the following flags.

printer/writeForm/text

Device has Text capability.

printer/writeForm/graphics

Device has Graphics capability.

printer/writeForm/ocr

Device has OCR capability.

printer/writeForm/micr

Device has MICR capability.

printer/writeForm/msf

Device has MSF capability.

printer/writeForm/barcode

Device has Barcode capability.

printer/writeForm/stamp

Device has stamping capability.

printer/extents

Specifies whether the device is able to measure the inserted media, as a combination of the following flags.

printer/extents/horizontal

Device has horizontal size detection capability.

printer/extents/vertical

Device has vertical size detection capability.

printer/control

Specifies the manner in which media can be controlled, as a combination of the following flags.

printer/control/eject

Device can eject media.

printer/control/perforate

Device can perforate media.

printer/control/cut

Device can cut media.

printer/control/skip

Device can skip to mark.

printer/control/flush

Device can be sent data that is buffered internally, and flushed to the printer on request.

printer/control/retract

Device can retract media under client control.

printer/control/stack

Device can stack media items before ejecting as a bundle.

printer/control/partialCut

Device can partially cut the media.

printer/control/alarm

Device can ring a bell, beep or otherwise sound an audible alarm.

printer/control/pageForward

Capability to turn one page forward.

printer/control/pageBackward

Capability to turn one page backward.

printer/control/turnMedia

Device can turn inserted media.

printer/control/stamp

Device can stamp on media.

printer/control/park

Device can park a document into the parking station.

printer/control/expel

Device can expel media out of the exit slot.

printer/control/ejectToTransport

Device can move media to a position on the transport just behind the exit slot.

printer/control/rotate180

Device can rotate media 180 degrees in the printing plane.

printer/control/clearBuffer

The Service Provider can clear buffered data.

printer/maxMediaOnStacker

Specifies the maximum number of media items that the stacker can hold (zero if not available).

printer/acceptMedia

Specifies whether the device is able to accept media while no execute command is running that is waiting explicitly for media to be inserted.

printer/multiPage

Specifies whether the device is able to support multiple page print jobs.

printer/paperSources

Specifies the Paper sources available for this printer as a combination of the following flags

printer/paperSources/upper

Indicates an upper paper source is available; devices with only one paper supply must indicate upper as being available.

printer/paperSources/lower

Indicates a lower paper source is available.

printer/paperSources/external

Indicates an external paper source (such as envelope tray or single sheet feed) is available.

printer/paperSources/aux

An auxiliary paper source is available.

printer/paperSources/aux2

A second auxiliary paper source is available.

printer/paperSources/park

A parking station is available.

printer/mediaTaken

Specifies whether the device is able to detect when the media is taken from the exit slot. If false, the [Printer.MediaTakenEvent](#) event is not fired.

printer/retractBins

Specifies the number of retract bins (zero if not supported).

printer/maxRetract

An array of the length [retractBins](#) with the maximum number of media items that each retract bin can hold (one count for each supported bin, starting from zero for bin number one to retractBins - 1 for bin number retractBins). This will be omitted if there are no retract bins.

printer/imageType

Specifies the image format supported by this device, as a combination of following flags.

printer/imageType/tif

The device can return scanned images in TIFF 6.0 format.

printer/imageType/wmf

The device can return scanned images in WMF (Windows Metafile) format.

printer/imageType/bmp

The device can return scanned images in Windows BMP format.

printer/imageType/jpg

The device can return scanned images in JPG format.

printer/frontImageColorFormat

Specifies the front image color formats supported by this device, as a combination of following flags.

printer/frontImageColorFormat/binary

The device can return scanned images in binary (image contains two colors, usually the colors black and white).

printer/frontImageColorFormat/grayscale

The device can return scanned images in gray scale (image contains multiple gray colors).

printer/frontImageColorFormat/full

The device can return scanned images in full color (image contains colors like red, green, blue etc.).

printer/backImageColorFormat

Specifies the back image color formats supported by this device, as a combination of following flags.

printer/backImageColorFormat/binary

The device can return scanned images in binary (image contains two colors, usually the colors black and white).

printer/backImageColorFormat/grayScale

The device can return scanned images in gray scale (image contains multiple gray colors).

printer/backImageColorFormat/full

The device can return scanned images in full color (image contains colors like red, green, blue etc.).

printer/codelineFormat

Specifies the code line (MICR data) formats supported by this device, as a combination of following flags.

printer/codelineFormat/cmc7

The device can read CMC7 code lines.

printer/codelineFormat/e13b

The device can read E13B code lines.

printer/codelineFormat/ocr

The device can read code lines using Optical Character Recognition.

printer/imageSource

Specifies the source for the read image command supported by this device, as a combination of the following flags.

printer/imageSource/imageFront

The device can scan the front image of the document.

printer/imageSource/imageBack

The device can scan the back image of the document.

printer/imageSource/codeLine

The device can recognize the code line.

printer/dispensePaper

Specifies whether the device is able to dispense paper.

printer/osPrinter

Specifies the name of the default logical operating system printer that is associated with this Service Provider. Clients should use this printer name to generate native printer files to be printed through the [Printer.PrintRawFile](#) command. This value will be omitted if the Service Provider does not support the *Printer.PrintRawFile* command.

printer/mediaPresented

Specifies whether the device is able to detect when the media is presented to the user for removal. If true, the [Printer.MediaPresentedEvent](#) event is fired. If false, the [Printer.MediaPresentedEvent](#) event is not fired.

printer/autoRetractPeriod

Specifies the number of seconds before the device will automatically retract the presented media. If the command that generated the media is still active when the media is automatically retracted, the command will complete with an error. If the device does not retract media automatically this value will be zero.

printer/retractToTransport

Specifies whether the device is able to retract the previously ejected media to the transport.

printer/coercivityType

Specifies the form write modes supported by this device, as a combination of the following flags.

printer/coercivityType/low

This device can write the magnetic stripe by low coercivity mode.

printer/coercivityType/high

This device can write the magnetic stripe by high coercivity mode.

printer/coercivityType/auto

The Service Provider or the device is capable of automatically determining whether low or high coercivity magnetic stripe should be written.

printer/controlPassbook

Specifies how the passbook can be controlled with the [Printer.ControlPassbook](#) command, as a combination of the following flags.

printer/controlPassbook/turnForward

The device can turn forward multiple pages of the passbook.

printer/controlPassbook/turnBackward

The device can turn backward multiple pages of the passbook.

printer/controlPassbook/closeForward

The device can close the passbook forward.

printer/controlPassbook/closeBackward

The device can close the passbook backward.

printer/printSides

Specifies on which sides of the media this device can print as one of the following values.

- notSupp - The device is not capable of printing on any sides of the media.
- single - The device is capable of printing on one side of the media.
- dual - The device is capable of printing on two sides of the media.

SensorsAndIndicators

Capability information for XFS4IoT services implementing the Sensors and Indicators interface. This will be omitted if the Sensors and Indicators interface is not supported.

SensorsAndIndicators/sensorType

Specifies the type of sensors and indicators supported by this device.

SensorsAndIndicators/sensorType/sensors

The device supports input sensors.

SensorsAndIndicators/sensorType/doors

The device supports door sensors.

SensorsAndIndicators/sensorType/indicators

The device supports indicators.

SensorsAndIndicators/sensorType/auxiliary

The device supports auxiliary indicators.

SensorsAndIndicators/sensorType/guidelights

The device supports guidance lights.

SensorsAndIndicators/sensorType/operatorSwitch

Specifies the Operator switch.

SensorsAndIndicators/sensorType/tamperSensor

Specifies the Tamper sensor.

SensorsAndIndicators/sensorType/intTamperSensor

Specifies the internal Tamper sensor.

SensorsAndIndicators/sensorType/seismicSensor

Specifies the Seismic sensor.

SensorsAndIndicators/sensorType/heatSensor

Specifies the heat sensor.

SensorsAndIndicators/sensorType/proximitySensor

Specifies the proximity sensor.

SensorsAndIndicators/sensorType/ambientLightSensor

Specifies the ambient light sensor.

SensorsAndIndicators/sensorType/enhancedAudioSensor

Specifies whether the Audio Jack is present, and if so, which modes it supports.

SensorsAndIndicators/sensorType/bootSwitchSensor

Specifies the boot switch sensor.

SensorsAndIndicators/sensorType/displaySensor

Specifies the Consumer Display.

SensorsAndIndicators/sensorType/operatorCallButtonSensor

Specifies whether the Operator Call Button is available. The Operator Call Button does not actually call the operator but just sends a signal to the client.

SensorsAndIndicators/sensorType/handsetSensor

Specifies whether the Handset is present, and if so, which modes it supports.

SensorsAndIndicators/sensorType/generalInputPort

Specifies whether the vendor dependent General-Purpose Input Ports are available. This value is an array and each index represents one General-Purpose Input Port.

SensorsAndIndicators/sensorType/headsetMicrophoneSensor

Specifies whether the Microphone Jack is present, and if so, which modes it supports.

SensorsAndIndicators/sensorType/fasciaMicrophoneSensor

Specifies whether a Fascia Microphone (for public audio input) is present.

SensorsAndIndicators/sensorType/cabinetDoor

Specifies whether at least one Cabinet Doors is available, and if so, which states they can take.

SensorsAndIndicators/sensorType/safeDoor

Specifies whether the safe Door is available, and if so, which states it can take.

SensorsAndIndicators/sensorType/vandalShield

Specifies whether the Vandal Shield is available, and if so, which states it can take.

SensorsAndIndicators/sensorType/frontCabinet

Specifies whether at least one Front Cabinet Door is available, and if so, which states they can take.

SensorsAndIndicators/sensorType/rearCabinet

Specifies whether at least one rear Cabinet Door is available, and if so, which states they can take.

SensorsAndIndicators/sensorType/leftCabinet

Specifies whether at least one left Cabinet Door is available, and if so, which states they can take.

SensorsAndIndicators/sensorType/rightCabinet

Specifies whether at least one right Cabinet Door is available, and if so, which states they can take.

SensorsAndIndicators/sensorType/openCloseIndicator

Specifies whether the Open/Closed Indicator is available.

SensorsAndIndicators/sensorType/fasciaLight

Specifies whether the fascia light is available.

SensorsAndIndicators/sensorType/audio

Specifies whether the Audio Indicator device is available.

SensorsAndIndicators/sensorType/heating

Specifies whether the internal Heating device is available.

SensorsAndIndicators/sensorType/consumerDisplayBacklight

Specifies whether the Consumer Display Backlight is available.

SensorsAndIndicators/sensorType/signageDisplay

Specifies whether the Signage Display is available.

SensorsAndIndicators/sensorType/transactionIndicator

Specifies whether the Transaction Indicators are available as an array. Each index of this array represents one Transaction Indicator .

SensorsAndIndicators/sensorType/generalOutputPort

Specifies whether the vendor dependent General-Purpose Output Ports are available. This value is an array and each index represents one General-Purpose Output Port.

SensorsAndIndicators/sensorType/volume

Specifies whether the Volume Control is available, and if so, the increment/decrement value recommended by the vendor.

SensorsAndIndicators/sensorType/UPS

Specifies whether the UPS device is available, and if so, which states it can take.

SensorsAndIndicators/sensorType/remoteStatusMonitor

Specifies whether the Remote Status Monitor device is available.

SensorsAndIndicators/sensorType/audibleAlarm

Specifies whether the Audible Alarm device is available.

SensorsAndIndicators/sensorType/enhancedAudioControl

Specifies whether the Enhanced Audio Controller is available, and if so, which modes it supports.

SensorsAndIndicators/sensorType/enhancedMicrophoneControlState

Specifies whether the Enhanced Microphone Controller is available, and if so, which modes it supports.

SensorsAndIndicators/sensorType/microphoneVolume

Specifies whether the Microphone Volume Control is available, and if so, the increment/decrement value recommended by the vendor.

SensorsAndIndicators/sensorType/autoStartupMode

Specifies which mode of the auto start-up control is supported.

SensorsAndIndicators/sensorType/guideLights

Available guidelights.

SensorsAndIndicators/sensorType/guideLights/cardReader

Card Unit Guidelight.

SensorsAndIndicators/sensorType/guideLights/cardReader/flashRate

Indicates the guidelight flash rate. The following values are possible: "notAvailable": The light indicator is not available. "off": The light can be turned off. "slow": The light can blink slowly. "medium": The light can blink medium frequency. "quick": The light can blink quickly. "continuous": The light can be continuous (steady).

SensorsAndIndicators/sensorType/guideLights/cardReader/colour

Indicates the guidelight colour. The following values are possible: "defaultColor": The light indicator is not available. "red": The light can be red. "green": The light can be green. "yellow": The light can be yellow. "blue": The light can be blue. "cyan": The light can be cyan. "magenta": The light can be magenta. "white": The light can be white.

SensorsAndIndicators/sensorType/guideLights/cardReader/direction

Indicates the guidelight direction. The following values are possible: "entry": The light can indicate entry. "exit": The light can indicate exit.

SensorsAndIndicators/sensorType/guideLights/cardReader/position

Indicates the guidelight position. The following values are possible: "default": The default position. "left": The left position. "right": The right position. "center": The center position. "top": The top position. "bottom": The bottom position. "front": The front position. "rear": The rear position.

SensorsAndIndicators/sensorType/guideLights/pinPad

Pin Pad Guidelight.

SensorsAndIndicators/sensorType/guideLights/notesDispenser

Notes Dispenser Guidelight.

SensorsAndIndicators/sensorType/guideLights/coinDispenser

Coin Dispenser Guidelight.

SensorsAndIndicators/sensorType/guideLights/receiptPrinter

Receipt Printer Guidelight.

SensorsAndIndicators/sensorType/guideLights/passbookPrinter

Passbook Printer Guidelight.

SensorsAndIndicators/sensorType/guideLights/EnvelopeDepository

Envelope Depository Guidelight.

SensorsAndIndicators/sensorType/guideLights/chequeUnit

Cheque Unit Guidelight.

SensorsAndIndicators/sensorType/guideLights/billAcceptor

Bill Acceptor Guidelight.

SensorsAndIndicators/sensorType/guideLights/envelopeDispenser

Envelope Dispenser Guidelight.

SensorsAndIndicators/sensorType/guideLights/documentPrinter

Document Printer Guidelight.

SensorsAndIndicators/sensorType/guideLights/coinAcceptor

Coin Acceptor Guidelight.

SensorsAndIndicators/sensorType/guideLights/scanner

scanner Guidelight.

SensorsAndIndicators/sensorType/guideLights/contactless

Contactless Guidelight.

SensorsAndIndicators/sensorType/guideLights/cardUnit2

Card Unit 2 Guidelight.

SensorsAndIndicators/sensorType/guideLights/notesDispenser2

Notes Dispenser 2 Guidelight.

SensorsAndIndicators/sensorType/guideLights/billAcceptor2

Bill Acceptor 2 Guidelight.

SensorsAndIndicators/sensorType/guideLights/vendorDependent

Vendor Dependent Guidelight.

SensorsAndIndicators/sensorType/guideLights/flashRate

Indicates the guidelight flash rate. The following values are possible: "notAvailable": The light indicator is not available. "off": The light can be turned off. "slow": The light can blink slowly. "medium": The light can blink medium frequency. "quick": The light can blink quickly. "continuous":The light can be continuous (steady).

SensorsAndIndicators/sensorType/guideLights/colour

Indicates the guidelight colour. The following values are possible: "defaultColor": The light indicator is not available. "red": The light can be red. "green": The light can be green. "yellow": The light can be yellow. "blue": The light can be blue. "cyan": The light can be cyan. "magenta": The light can be magenta. "white": The light can be white.

SensorsAndIndicators/sensorType/guideLights/direction

Indicates the guidelight direction. The following values are possible: "entry": The light can indicate entry. "exit": The light can indicate exit.

SensorsAndIndicators/sensorType/guideLights/position

Indicates the guidelight position. The following values are possible: "default": The default position. "left": The left position. "right": The right position. "center": The center position. "top": The top position. "bottom": The bottom position. "front": The front position. "rear": The rear position.

cardEmbosser

Capability information for XFS4IoT services implementing the CardEmbosser interface. This will be omitted if the CardEmbosser interface is not supported.

cardEmbosser/compareMagneticStripe

Indicates whether the card embosser has capability of comparing magnetic stripe contents (* true*) as a prerequisite for an encoding or embossing operation.

cardEmbosser/magneticStripeRead

Indicates whether the card embosser has magnetic stripe reading capability.

cardEmbosser/magneticStripeWrite

Indicates whether the card embosser has magnetic stripe writing capability.

cardEmbosser/chipIO

Indicates whether the card embosser has smart card updating capability.

cardEmbosser/chipProtocol

Specifies the chip card protocols that are supported by the Service Provider as a combination of the following:

cardEmbosser/chipProtocol/notSupported

The card embosser can not handle chip cards.

cardEmbosser/chipProtocol/chipT0

The card embosser can handle the T=0 protocol.

cardEmbosser/chipProtocol/chipT1

The card embosser can handle the T=1 protocol.

cardEmbosser/chipProtocol/chipProtocolNotRequired

The card embosser is capable of communicating with a chip card without requiring the client to specify any protocol.

cardEmbosser/charSupport

Specifies the character sets, in addition to single byte ASCII, that is supported by the Service Provider

A Service Provider can support ONLY ASCII forms or can support BOTH ASCII and UNICODE forms. A Service Provider cannot support UNICODE forms without also supporting ASCII forms.

This field will be set to a combination of the following:

cardEmbosser/charSupport/ascii

ASCII is supported for XFS forms.

cardEmbosser/charSupport/unicode

UNICODE is supported for XFS forms.

cardEmbosser/type

Specifies whether the card embosser has a card embossing capability and/or a card printing capability. This field will be set to a combination of the following:

cardEmbosser/type/emboss

The card embosser supports embossing data on cards.

cardEmbosser/type/print

The card embosser supports printing data on cards.

barcodeReader

Capability information for XFS4IoT services implementing the BarcodeReader interface. This will be omitted if the BarcodeReader interface is not supported.

barcodeReader/canFilterSymbologies

Specifies whether the device is capable of discriminating between the presented barcode symbologies such that only the desired symbologies are recognized/reported

barcodeReader/symbologies

Specifies the barcode symbologies readable by the scanner. This will be omitted if the supported barcode symbologies can not be determined.

barcodeReader/symbologies/ean128

GS1-128

barcodeReader/symbologies/ean8

EAN-8

barcodeReader/symbologies/ean8_2

EAN-8 with 2 digit add-on

barcodeReader/symbologies/ean8_5

EAN-8 with 5 digit add-on

barcodeReader/symbologies/ean13

EAN13

barcodeReader/symbologies/ean13_2

EAN-13 with 2 digit add-on

barcodeReader/symbologies/ean13_5

EAN-13 with 5 digit add-on

barcodeReader/symbologies/jan13

jan-13

barcodeReader/symbologies/upcA

UPC-A

barcodeReader/symbologies/upcE0

UPC-E

barcodeReader/symbologies/upcE0_2

UPC-E with 2 digit add-on

barcodeReader/symbologies/upcE0_5

UPC-E with 5 digit add-on

barcodeReader/symbologies/upcE1

UPC-E with leading 1

barcodeReader/symbologies/upcE1_2

UPC-E with leading 1and 2 digit add-on

barcodeReader/symbologies/upcE1_5

UPC-E with leading 1and 5 digit add-on

barcodeReader/symbologies/upcA_2

UPC-A with2 digit add-on

barcodeReader/symbologies/upcA_5

UPC-A with 5 digit add-on

barcodeReader/symbologies/codabar

CODABAR (NW-7)

barcodeReader/symbologies/itf

Interleaved 2 of 5 (ITF)

barcodeReader/symbologies/code11

CODE 11 (USD-8)

barcodeReader/symbologies/code39

CODE 39

barcodeReader/symbologies/code49

CODE 49

barcodeReader/symbologies/code93

CODE 93

barcodeReader/symbologies/code128

CODE 128

barcodeReader/symbologies/msi

MSI

barcodeReader/symbologies/plessey

PLESSEY

barcodeReader/symbologies/std2Of5

STANDARD 2 of 5 (INDUSTRIAL 2 of 5 also)

barcodeReader/symbologies/std2Of5Iata

STANDARD 2 of 5 (IATA Version)

barcodeReader/symbologies/pdf417

PDF-417

barcodeReader/symbologies/microPdf417

MICROPDF-417

barcodeReader/symbologies/dataMatrix

GS1 DataMatrix

barcodeReader/symbologies/maxicode

MAXICODE

barcodeReader/symbologies/codeOne

CODE ONE

barcodeReader/symbologies/channelCode

CHANNEL CODE

barcodeReader/symbologies/telepenOriginal

Original TELEPEN

barcodeReader/symbologies/telepenAim

AIM version of TELEPEN

barcodeReader/symbologies/rss

GS1 DataBar™

barcodeReader/symbologies/rssExpanded

Expanded GS1 DataBar™

barcodeReader/symbologies/rssRestricted

Restricted GS1 DataBar™

barcodeReader/symbologies/compositeCodeA

Composite Code A Component

barcodeReader/symbologies/compositeCodeB

Composite Code B Component

barcodeReader/symbologies/compositeCodeC

Composite Code C Component

barcodeReader/symbologies/posiCodeA

Posicode Variation A

barcodeReader/symbologies/posiCodeB

Posicode Variation B

barcodeReader/symbologies/triopticCode39

Trioptic Code 39

barcodeReader/symbologies/codablockF

Codablock F

barcodeReader/symbologies/code16K

Code 16K

barcodeReader/symbologies/qrCode

QR Code

barcodeReader/symbologies/aztec

Aztec Codes

barcodeReader/symbologies/ukPost

UK Post

barcodeReader/symbologies/planet

US Postal Planet

barcodeReader/symbologies/postnet

US Postal Postnet

barcodeReader/symbologies/canadianPost

Canadian Post

barcodeReader/symbologies/netherlandsPost

Netherlands Post

barcodeReader/symbologies/australianPost

Australian Post

barcodeReader/symbologies/japanesePost

Japanese Post

barcodeReader/symbologies/chinesePost

Chinese Post

barcodeReader/symbologies/koreanPost

Korean Post

biometric

Capability information for XFS4IoT services implementing the Biometrics interface. This will be omitted if the Biometrics interface is not supported.

biometric/type

Specifies the type of biometric device as a combination.

biometric/type/facialFeatures

The biometric device supports facial recognition scanning.

biometric/type/voice

The biometric device supports voice recognition.

biometric/type/fingerprint

The biometric device supports fingerprint scanning.

biometric/type/fingerVein

The biometric device supports finger vein scanning.

biometric/type/iris

The biometric device supports iris scanning.

biometric/type/retina

The biometric device supports retina scanning.

biometric/type/handGeometry

The biometric device supports hand geometry scanning.

biometric/type/thermalFace

The biometric device supports thermal face image scanning.

biometric/type/thermalHand

The biometric device supports thermal hand image scanning.

biometric/type/palmVein

The biometric device supports palm vein scanning.

biometric/type/signature

The biometric device supports signature scanning.

biometric/compound

Specifies whether the biometric device is part of a compound device

biometric/maxCapture

Specifies the maximum number of times that the device can attempt to capture biometric data during a [Biometric.Read](#) command. If this is zero then the device or service provider determines how many captures will be attempted.

biometric/templateStorage

Specifies the storage space that is reserved on the device for the storage of templates in bytes. This will be set to zero if not reported or unknown.

biometric/dataFormats

Specifies the supported biometric raw data and template data formats reported

biometric/dataFormats/isoFid

Raw ISO FID format

biometric/dataFormats/isoFmd

ISO FMD template format

biometric/dataFormats/ansiFid

Raw ANSI FID format

biometric/dataFormats/ansiFmd

ANSI FMD template format

biometric/dataFormats/qso

Raw QSO image format

biometric/dataFormats/wso

WSQ image format

biometric/dataFormats/reservedRaw1

Reserved for a vendor-defined Raw format.

biometric/dataFormats/reservedTemplate1

Reserved for a vendor-defined Template format.

biometric/dataFormats/reservedRaw2

Reserved for a vendor-defined Raw format.

biometric/dataFormats/reservedTemplate2

Reserved for a vendor-defined Template format.

biometric/dataFormats/reservedRaw3

Reserved for a vendor-defined Raw format.

biometric/dataFormats/reservedTemplate3

Reserved for a vendor-defined Template format.

biometric/encryptionAlgorithm

Supported encryption algorithms or cryptNone if no encryption algorithms

biometric/encryptionAlgorithm/ecb

Triple DES with Electronic Code Book.

biometric/encryptionAlgorithm/cbc

Triple DES with Cipher Block Chaining

biometric/encryptionAlgorithm/cfb

Triple DES with Cipher Feed Back.

biometric/encryptionAlgorithm/rsa

RSA Encryption.

biometric/storage

Indicates whether or not biometric template data can be stored securely or none if Biometric template data is not stored in the device

biometric/storage/secure

Biometric template data is securely stored as encrypted data.

biometric/storage/clear

Biometric template data is stored unencrypted in the device.

biometric/persistenceModes

Specifies which data persistence modes can be set using the [Biometric.SetDataPersistence](#) command. This applies specifically to the biometric data that has been captured using the [Biometric.Read](#) command. A value of none indicates that persistence is entirely under device control and cannot be set.

biometric/persistenceModes/persist

Biometric data captured using the [Biometric.Read](#) command can persist until all sessions are closed, the device is power failed or rebooted, or the [Biometric.Read](#) command is requested again. This captured biometric data can also be explicitly cleared using the [Biometric.Clear](#) or [Biometric.Reset](#) commands.

biometric/persistenceModes/clear

Captured biometric data will not persist. Once the data has been either returned in the [Biometric.Read](#) command or used by the [Biometric.Match](#) command, then the data is cleared from the device.

biometric/matchSupported

Specifies if matching is supported using the [Biometric.Match](#) and/or [Biometric.SetMatch](#) command. This will be one of the following values:

- None - The device does not support matching.
- StoredMatch - The device scans biometric data using the [Biometric.Read](#) command and stores it, then the scanned data can be compared with imported biometric data using the [Biometric.Match](#) command
- CombinedMatch - The device scans biometric data and performs a match against imported biometric data as a single operation. The [Biometric.SetMatch](#) command must be called before the [Biometric.Read](#) command in order to set the matching criteria. Then the [Biometric.Match](#) command can be called to return the result

biometric/scanModes

Specifies the modes that the [Biometric.Read](#) command.

biometric/scanModes/scan

The [Biometric.Read](#) command can be used to scan data only, for example to enroll a user or collect data for matching in an external biometric system.

biometric/scanModes/match

The [Biometric.Read](#) command can be used to scan data for a match operation using the [Biometric.Match](#) command.

biometric/compareModes

Specifies the type of match operations. A value of none indicates that matching is not supported

biometric/compareModes/verify

The biometric data can be compared as a one to one verification operation.

biometric/compareModes/identity

The biometric data can be compared as a one to many identification operation

biometric/clearData

Specifies the type of data that can be cleared from storage using the [Biometric.Clear](#) or [Biometric.Reset](#) command as either none.

biometric/clearData/scannedData

Raw image data that has been scanned using the [Biometric.Read](#) command can be cleared

biometric/clearData/importedData

Template data that was imported using the [Biometric.Import](#) command can be cleared.

biometric/clearData/setMatchedData

Match criteria data that was set using the [Biometric.Match](#) command can be cleared.

Event Messages

None

[2.2.3 - Common.SetGuidanceLight](#)

This command is used to set the status of the devices guidance lights. This includes defining the flash rate, the color and the direction. When a client tries to use a color or direction that is not supported then the Service Provider will return the generic completionCode unsupportedData.

Command Message

| Payload | Type | Required |
|------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |

```
"guidLight": 0,      integer  
"command": {        object  
  "flashRate": "off",    string  
  "color": "default",   string  
  "direction": "entry"  string  
}  
}
```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

guidLight

Specifies the index of the guidance light to set as one of the values defined within the capabilities section:

command/flashRate

Indicates which flash rates are supported by the guidelight.

command/color

Indicates which colors are supported by the guidelight.

command/direction

Indicates which directions are supported by the guidelight. and it's an optional field

Completion Message

| Payload | Type | Required |
|---|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML | string | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

Event Messages

None

2.2.4 - Common.PowerSaveControl

This command activates or deactivates the power-saving mode. If the Service Provider receives another execute command while in power saving mode, the Service Provider automatically exits the power saving mode, and executes the requested command. If the Service Provider receives an information command while in power saving mode, the Service Provider will not exit the power saving mode.

Command Message

| Payload | Type | Required |
|------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |

```
"maxPowerSaveRecoveryTime": 0 integer
```

```
}
```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

maxPowerSaveRecoveryTime

Specifies the maximum number of seconds in which the device must be able to return to its normal operating state when exiting power save mode. The device will be set to the highest possible power save mode within this constraint. If usMaxPowerSaveRecoveryTime is set to zero then the device will exit the power saving mode.

Completion Message

| Payload | Type | Required |
|---|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional

information

Event Messages

- [Common.PowerSaveChangeEvent](#)

[2.2.5 - Common.SynchronizeCommand](#)

This command is used to reduce response time of a command (e.g. for synchronization with display) as well as to synchronize actions of the different device classes. This command is intended to be used only on hardware which is capable of synchronizing functionality within a single device class or with other device classes.

The list of execute commands which this command supports for synchronization is retrieved in the [synchronizableCommands](#) value of [Common.Capabilities](#).

This command is optional, i.e. any other command can be called without having to call it in advance. Any preparation that occurs by calling this command will not affect any other subsequent command. However, any subsequent execute command other than the one that was specified in the *command* input parameter will execute normally and may invalidate the pending synchronization. In this case the client should call [Common.SynchronizeCommand](#) again in order to start a synchronization.

Command Message

| Payload | Type | Required |
|---------------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "command": Add example to YAML, | string | |
| "cmdData": { | object | |
| } | | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

timeout but can be cancelled.

default: 0

command

The command name to be synchronized and executed next.

cmdData

A payload that represents the parameter that is normally associated with the command.

Completion Message

| Payload | Type | Required |
|---|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

Event Messages

None

2.2.6 - Common.SetTransactionState

This command allows the client to specify the transaction state, which the Service Provider can then utilize in order to optimize performance. After receiving this command, this Service Provider can perform the necessary processing to start or end the customer transaction. This command should be called for every Service Provider that could be used in a customer transaction. The transaction state applies to every session.

Command Message

| Payload | Type | Required |
|---------------------------------------|----------------|----------|
| { | | |
| "state": "active", | string | |
| "transactionID": Add example to YAML, | string | |
| "extra": [Add example to YAML] | array (string) | |
| } | | |

Properties

state

Specifies the transaction state. Following values are possible:

"active": A customer transaction is in progress.

"inactive": No customer transaction is in progress.

transactionID

Specifies a string which identifies the transaction ID. The value returned in this parameter is a client defined customer transaction identifier, which was previously set in the Common.SetTransactionState command

extra

A list of vendor-specific, or any other extended, transaction information. The information is set as a series of "key=value" strings. Each string is null-terminated, with the final string terminating with two null characters. An empty list may be indicated by either a NULL pointer or a pointer to two consecutive null characters

Completion Message

| Payload | Type | Required |
|---|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML | string | |
| } | | |

*Properties***completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

Event Messages

None

2.2.7 - Common.GetTransactionState

This command can be used to get the transaction state.

Command Message

| Payload | Type | Required |
|-----------------|---------|----------|
| { | | |
| "timeout": 5000 | integer | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|---------------------------------------|----------------|----------|
| { | | |
| "state": "active", | string | |
| "transactionID": Add example to YAML, | string | |
| "extra": [Add example to YAML] | array (string) | |
| } | | |

Properties

state

Specifies the transaction state. Following values are possible:

"active": A customer transaction is in progress.

"inactive": No customer transaction is in progress.

transactionID

Specifies a string which identifies the transaction ID. The value returned in this parameter is a client defined customer transaction identifier, which was previously set in the Common.SetTransactionState command

extra

A list of vendor-specific, or any other extended, transaction information. The information is set as a series of "key=value" strings. Each string is null-terminated, with the final string terminating with two null characters. An empty list may be indicated by either a NULL

pointer or a pointer to two consecutive null characters

Event Messages

None

2.2.8 - Common.GetCommandNonce

Get a nonce to be included in an Authorisation Token for a command that will be used to ensure end to end security.

The hardware will overwrite any existing stored Command nonce with this new value. The value will be stored for future authentication. Any Authorisation Token received will be compared with this stored nonce and if the Token doesn't contain the same nonce it will be considered invalid and rejected, causing the command that contains that Authentication Token to fail.

The nonce must match the algorithm used. For example, HMAC SHA256 means the nonce must be 256 bit/32 bytes.

Command Message

| Payload | Type | Required |
|-----------------|---------|----------|
| { | | |
| "timeout": 5000 | integer | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "CommandNonce": 646169ECDD0E440C2CECC8DDD7C27C22 | string | |
| } | | |

Properties**CommandNonce**

A nonce that should be included in the authorisation token in a command used to provide end to end protection.

The nonce will be given as HEX (upper case.)

Event Messages

None

2.3 - Unsolicited Messages**2.3.1 - Common.PowerSaveChangeEvent**

This service event specifies that the power save recovery time has changed.

| Payload | Type | Required |
|----------------------------|---------|----------|
| { | | |
| "powerSaveRecoveryTime": 0 | integer | |
| } | | |

Properties

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

powerSaveRecoveryTime

Specifies the actual number of seconds required by the device to resume its normal operational state. This value is zero if the device exited the power saving mode

2.3.2 - Common.DevicePositionEvent

This service event reports that the device has changed its position status.

| Payload | Type | Required |
|------------------------------------|--------|----------|
| { "Position": "inposition" } | string | |

Properties

Position

Position of the device

3 - Card Reader Interface

This chapter defines the Card Reader interface functionality and messages.

3.1 - Summary

This interface allows for the operation of the following categories of card readers:

- Motorized card reader/writer
- Swipe card reader (writing facilities only partially included)
- Dip card reader
- Latched dip card reader
- Contactless chip card readers
- Permanent chip card readers (each chip is accessed through a unique service)

Some motorized card reader/writers have parking stations inside and can place identification cards there. Once a card is in its parking station another card can be accepted by the card reader. Cards may only be moved out of a parking station if there is no other card present in the media read/write position, the chip I/O position, the transport, or the entry/exit slot.

The following tracks/chips and the corresponding international standards are taken into account in this document:

- Track 1 - ISO 7811
- Track 2 - ISO 7811
- Track 3 - ISO 7811 / ISO 4909
- Cash Transfer Card Track 1 - (JIS I: 8 bits/char) Japan
- Cash Transfer Card Track 3 - (JIS I: 8 bits/char) Japan
- Front Track 1 - (JIS II) Japan
- Watermark - Sweden
- Chip (contacted) - ISO 7816
- Chip (contactless) - ISO 10536, ISO 14443 and ISO 18092

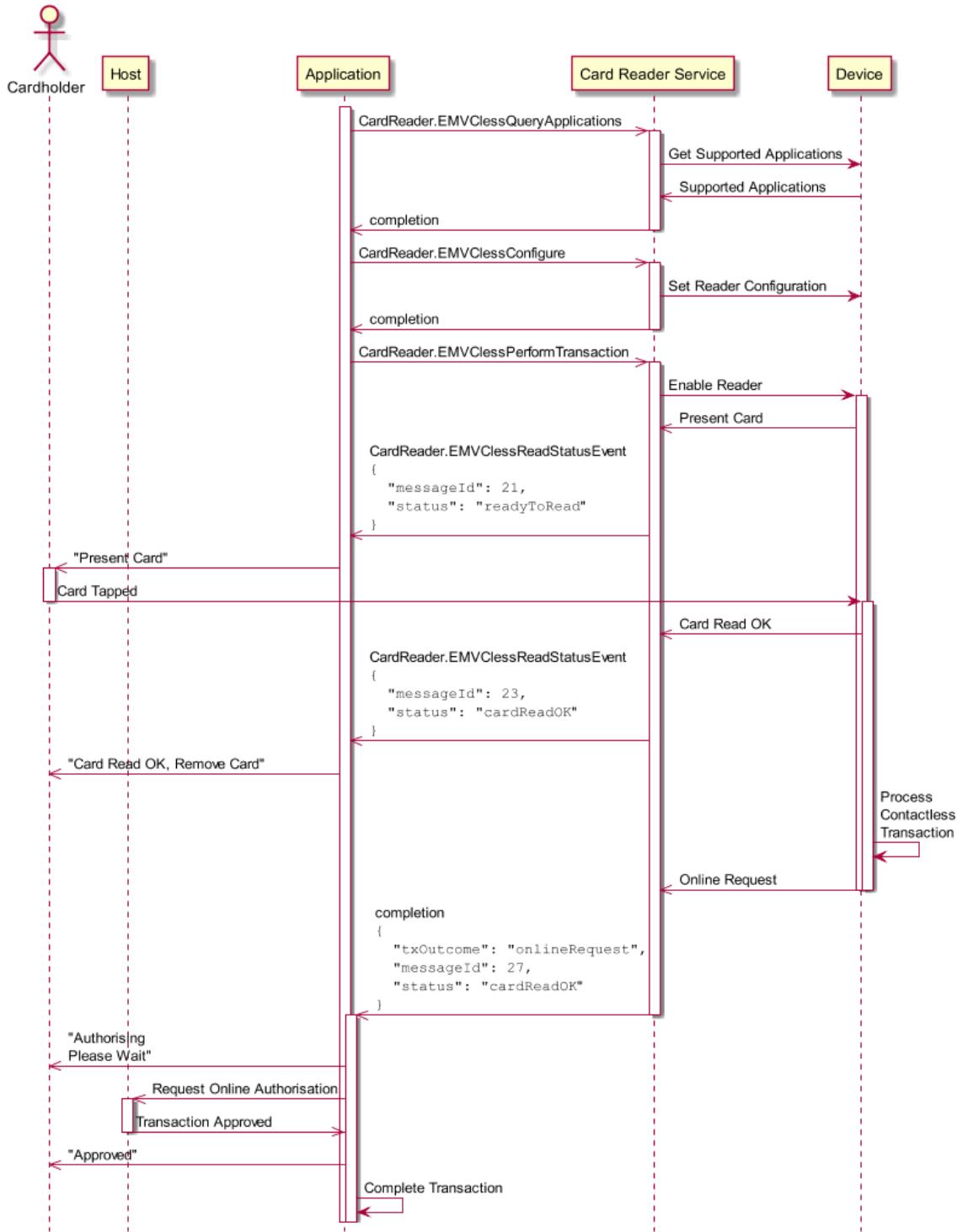
3.2 - General Information

3.2.1 - Intelligent Contactless Sequence Diagrams

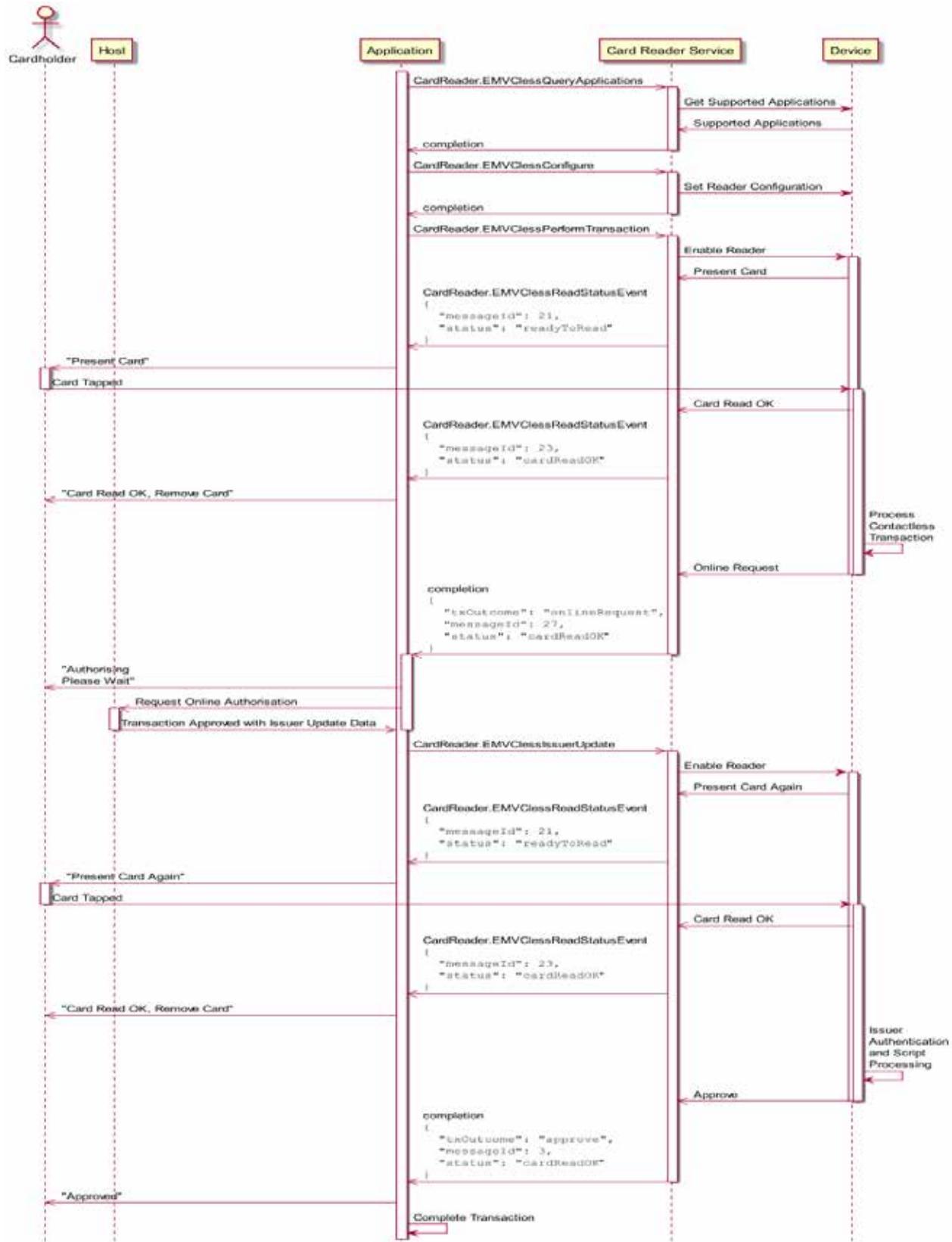
This section illustrates the sequence diagrams of EMV-like intelligent contactless transactions.

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

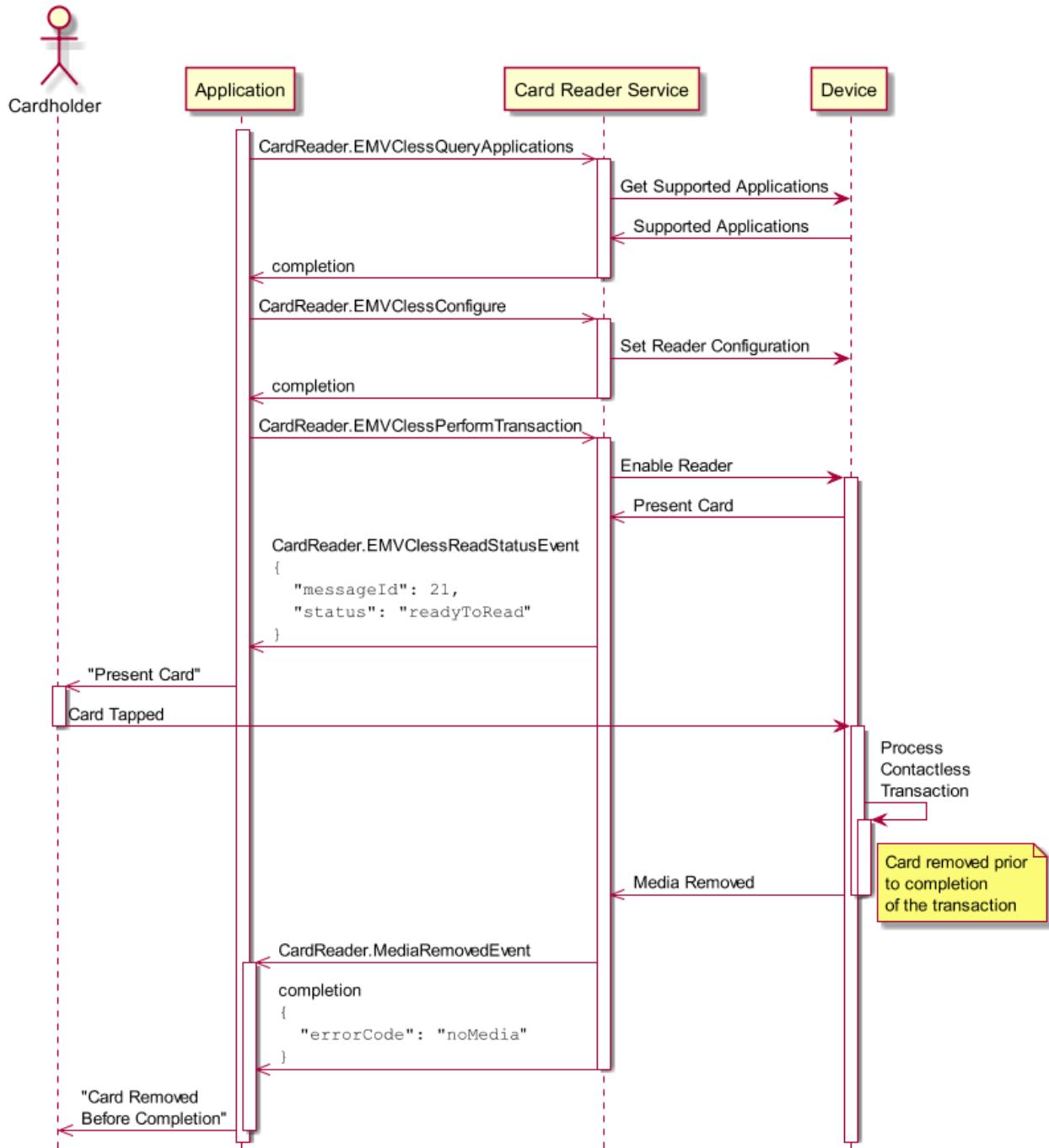
3.2.1.1 - Single Tap Transaction Without Issuer Update Processing



3.2.1.2 - Double Tap Transaction With Issuer Update Processing



3.2.1.3 - Card Removed Before Completion



3.3 - Command Messages

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

3.3.1 - CardReader.QueryIFMIdentifier

This command is used to retrieve the complete list of registration authority Interface Module (IFM) identifiers. The primary registration authority is EMVCo but other organizations are also supported for historical or local country requirements.

New registration authorities may be added in the future so applications should be able to handle the return of new (as yet undefined) IFM identifiers.

Command Message

| Payload | Type | Required |
|--------------------------------------|------|----------|
| { "timeout": 5000 integer } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "ifmAuthority": "emv", | string | |
| "ifmIdentifier": Add example to YAML | string | |
| } | | |

Properties

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

ifmAuthority

Specifies the IFM authority that issued the IFM identifier:

- emv - The Level 1 Type Approval IFM identifier assigned by EMVCo.
- europay - The Level 1 Type Approval IFM identifier assigned by Europay.
- visa - The Level 1 Type Approval IFM identifier assigned by VISA.
- giecb - The IFM identifier assigned by GIE Cartes Bancaires.

ifmIdentifier

The IFM Identifier of the chip card reader (or IFM) as assigned by the specified authority.

Event Messages

None

[3.3.2 - CardReader.EMVClessQueryApplications](#)

This command is used to retrieve the supported payment system applications available within an intelligent contactless card unit. The payment system application can either be identified by an AID or by the AID in combination with a Kernel Identifier. The Kernel Identifier has been introduced by the EMVCo specifications; see Reference [3].

Command Message

| Payload | Type | Required |
|---------|------|----------|
| { | | |

"timeout": 5000 integer

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

}

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "appData": [| array (object) | |
| "aid": Add example to YAML, | string | |
| "kernelIdentifier": Add example to YAML | string | |
|] | | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

appData

An array of application data objects which specifies a supported identifier (AID) and associated Kernel Identifier.

appData/aid

Contains the Base64 encoded payment system application identifier (AID) supported by the intelligent contactless card unit.

appData/kernelIdentifier

Contains the Base64 encoded Kernel Identifier associated with the *aid*. This data may be empty if the reader does not support Kernel Identifiers for example in the case of legacy approved contactless readers.

Event Messages

None

3.3.3 - CardReader.EjectCard

This command is only applicable to motor driven card readers and latched dip card readers.

For motorized card readers the default operation is that the card is driven to the exit slot from where the user can remove it. The card remains in position for withdrawal until either it is taken or another command is issued that moves the card.

For latched dip readers, this command causes the card to be unlatched (if not already unlatched), enabling removal.

After successful completion of this command, a [CardReader.MediaRemovedEvent](#) is generated to inform the application when the card is taken.

Command Message

| Payload | Type | Required |
|------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |

```
"ejectPosition": "transportPosition"  string
}
```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

ejectPosition

Specifies the destination of the card ejection for motorized card readers. Possible values are one of the following:

- `exitPosition` - The card will be transferred to the exit slot from where the user can remove it. In the case of a latched dip the card will be unlatched, enabling removal.
- `transportPosition` - The card will be transferred to the transport just behind the exit slot. If a card is already at this position then `success` will be returned. Another `CardReader.EjectCard` command is required with the `ejectPosition` set to `exitPosition` in order to present the card to the user for removal.

default: `exitPosition`

Completion Message

This event notifies the completion of the command and if successful includes the requested data.

| Payload | Type | Required |
|---|--------|----------|
| { | | |
| <code>"completionCode": "success"</code> , | string | |
| <code>"errorDescription": Add example to YAML,</code> | string | |
| <code>"errorCode": "mediaJam"</code> | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- mediaJam - The card is jammed. Operator intervention is required. A possible scenario is also when an attempt to retain the card was made during attempts to eject it. The retain bin is full; no more cards can be retained. The current card is still in the device.
- shutterFail - The open of the shutter failed due to manipulation or hardware error. Operator intervention is required.
- noMedia - No card is present.
- mediaRetained - The card has been retained during attempts to eject it. The device is clear and can be used.

Event Messages

- [CardReader.MediaRemovedEvent](#)
- [CardReader.RetainBinThresholdEvent](#)

[3.3.4 - CardReader.RetainCard](#)

The card is removed from its present position (card inserted into device, card entering, unknown position) and stored in the retain bin; applicable to motor-driven card readers only. The ID card unit sends a [CardReader.RetainBinThresholdEvent](#) if the storage capacity of the retain bin is reached. If the storage capacity has already been reached, and the command cannot be executed, an error is returned and the card remains in its present position.

Command Message

| Payload | Type | Required |
|------------------------------------|------|----------|
| { "timeout": 5000 } } | | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|---|----------|
| { "completionCode": "success", "errorDescription": Add example to YAML, "errorCode": "mediaJam", "count": 0, "position": "unknown" } | string string string integer string | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

information

errorCode

Specifies the error code if applicable. The following values are possible:

- mediaJam - The card is jammed. Operator intervention is required.
- noMedia - No card has been inserted. The [position](#) parameter has the value *unknown*.
- retainBinFull - The retain bin is full; no more cards can be retained. The current card is still in the device.
- shutterFail - The open of the shutter failed due to manipulation or hardware error. Operator intervention is required.

count

Total number of ID cards retained up to and including this operation, since the last [CardReader.ResetCount](#) command was executed.

position

Position of card; only relevant if card could not be retained. Possible positions:

- unknown - The position of the card cannot be determined with the device in its current state.
- present - The card is present in the reader.
- entering - The card is in the entering position (shutter).

Event Messages

- [CardReader.MediaRemovedEvent](#)
- [CardReader.RetainBinThresholdEvent](#)
- [CardReader.MediaRetainedEvent](#)

[3.3.5 - CardReader.ResetCount](#)

This function resets the present value for number of cards retained to zero. The function is possible for motor-driven card readers only.

The number of cards retained is controlled by the service and can be requested before resetting via [Common.Status](#).

Command Message

| Payload | Type | Required |
|---------------------------|---------|----------|
| { "timeout": 5000 } | integer | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|---|--------|----------|
| { "completionCode": "success", "errorDescription": Add example to YAML } | string | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

Event Messages

- [CardReader.RetainBinThresholdEvent](#)

[3.3.6 - CardReader.SetKey](#)

This command is used for setting the DES key that is necessary for operating a CIM86 module. The command must be executed before the first read command is issued to the card reader.

Command Message

| Payload | Type | Required |
|---------------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "keyValue": Add example to YAML | string | |
| } | | |

[Properties](#)

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

keyValue

Contains the Base64 encoded payment containing the CIM86 DES key. This key is supplied by the vendor of the CIM86 module.

Completion Message

This event notifies the completion of the command and if successful includes the requested data.

| Payload | Type | Required |
|---------|------|----------|
|---------|------|----------|

```
{  
  "completionCode": "success",           string  
  "errorDescription": Add example to YAML,  string  
  "errorCode": "invalidKey"             string  
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- invalidKey - The key does not fit to the security module.

Event Messages

None

[3.3.7 - CardReader.ReadRawData](#)

For motor driven card readers, the card unit checks whether a card has been inserted. If so, all specified tracks are read immediately. If reading the chip is requested, the chip will be contacted and reset and the ATR (Answer To Reset) data will be read. When this command completes the chip will be in contacted position. This command can also be used for an explicit cold reset of a previously contacted chip.

This command should only be used for user cards and should not be used for permanently connected chips.

If no card has been inserted, and for all other categories of card readers, the card unit waits for the period of time specified in the call for a card to be either inserted or pulled through. The next step is trying to read all tracks specified.

The [CardReader.InsertCardEvent](#) will be generated when there is no card in the card reader and the device is ready to accept a card. In addition to that, a security check via a security module (i.e. MM, CIM86) can be requested. If the security check fails however this should not stop valid data being returned. The response *securityFail* will be returned if the command specifies only security data to be read and the security check could not be executed, in all other cases *ok* will be returned with the data field of the output parameter set to the relevant value including *hardwareError*.

For non-motorized Card Readers which read track data on card exit, the *invalidData* error code is returned when a call to is made to read both track data and chip data.

If the card unit is a latched dip unit then the device will latch the card when the chip card will be read, i.e. [chip](#) is specified (see below). The card will remain latched until a call to [CardReader.EjectCard](#) is made.

For contactless chip card readers a collision of two or more card signals may happen. In this case, if the device is not able to pick the strongest signal, *errorCardCollision* will be returned.

Command Message

| Payload | Type | Required |
|---|---------|----------|
| { | | |
| " timeout ": 5000, | integer | |
| " track1<td>boolean</td><td></td> | boolean | |
| " track2<td>boolean</td><td></td> | boolean | |
| " track3<td>boolean</td><td></td> | boolean | |
| " chip<td>boolean</td><td></td> | boolean | |
| " security<td>boolean</td><td></td> | boolean | |
| " fluxInactive<td>boolean</td><td></td> | boolean | |
| " watermark<td>boolean</td><td></td> | boolean | |
| " memoryChip<td>boolean</td><td></td> | boolean | |
| " track1Front<td>boolean</td><td></td> | boolean | |
| " frontImage<td>boolean</td><td></td> | boolean | |

```
"backImage": false,    boolean  
"track1JIS": false,   boolean  
"track3JIS": false,   boolean  
"ddi": false          boolean  
}
```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

track1

Track 1 of the magnetic stripe will be read.

default: false

track2

Track 2 of the magnetic stripe will be read.

default: false

track3

Track 3 of the magnetic stripe will be read.

default: false

chip

The chip will be read.

default: false

security

A security check will be performed.

default: false

fluxInactive

If the Flux Sensor is programmable it will be disabled in order to allow chip data to be read on cards which have no magnetic stripes.

default: false

watermark

The Swedish Watermark track will be read.

default: false

memoryChip

The memory chip will be read.

default: false

track1Front

Track 1 data is read from the magnetic stripe located on the front of the card. In some countries this track is known as JIS II track.

default: false

frontImage

The front image of the card will be read in Base64 PNG format.

default: false

backImage

The back image of the card will be read in Base64 PNG format.

default: false

track1JIS

Track 1 of Japanese cash transfer card will be read. In some countries this track is known as JIS I track 1 (8bits/char).

default: false

track3JIS

Track 3 of Japanese cash transfer card will be read. In some countries this track is known as JIS I track 1 (8bits/char).

default: false

ddi

Dynamic Digital Identification data of the magnetic stripe will be read.

default: false

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "mediaJam", | string | |
| "track1": { | object | |
| "status": "ok", | string | |
| "data": "QmFzZTY0IGVuY29kZWQgZGF0YQ==" | string | |
| }, | | |
| "track2": { | object | |
| See track1 properties. | | |
| }, | | |

"track3": { object

See [track1](#) properties.

},

"chip": [{ array (object)

See [track1](#) properties.

}],

"security": { object

"status": "ok", string

"data": "readLevel1" string

},

"watermark": { object

See [track1](#) properties.

},

"memoryChip": { object

"status": "ok", string

"data": "chipT0" string

},

"track1Front": { object

See [track1](#) properties.

},

"frontImage": { object

See [track1](#) properties.

},

"backImage": { object

See [track1](#) properties.

},

"track1JIS": { object

See [track1](#) properties.

},

"track3JIS": { object

See [track1](#) properties.

},

"ddi": { object

See [track1](#) properties.

}

}

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- mediaJam - The card is jammed. Operator intervention is required.
- shutterFail - The open of the shutter failed due to manipulation or hardware error. Operator intervention is required.
- noMedia - The card was removed before completion of the read action (the event [CardReader.MediaInsertedEvent](#) has been generated). For motor driven devices, the read is disabled; i.e. another command has to be issued to enable the reader for card entry.
- invalidMedia - No track or chip found; card may have been inserted or pulled through the wrong way.
- cardTooShort - The card that was inserted is too short. When this error occurs the

card remains at the exit slot.

- cardTooLong - The card that was inserted is too long. When this error occurs the card remains at the exit slot.
- securityFail - The security module failed reading the cards security sign.
- cardCollision - There was an unresolved collision of two or more contactless card signals.

track1

Contains the data read from track 1.

track1/status

The status values applicable to all data sources. Possible values are:

- ok - The data is OK.
- dataMissing - The track/chip/memory chip is blank.
- dataInvalid - The data contained on the track/chip/memory chip is invalid. This will typically be returned when [data](#) reports *badReadLevel* or *dataInvalid*.
- dataTooLong - The data contained on the track/chip/memory chip is too long.
- dataTooShort - The data contained on the track/chip/memory chip is too short.
- dataSourceNotSupported - The data source to read from is not supported by the Service Provider.
- dataSourceMissing - The data source to read from is missing on the card, or is unable to be read due to a hardware problem, or the module has not been initialized. For example, this will be returned on a request to read a Memory Card and the customer has entered a magnetic card without associated memory chip. This will also be reported when *data* reports *noData*, *notInitialized* or *hardwareError*. This will also be reported when the image reader could not create a BMP file due to the state of the image reader or due to a failure.

track1/data

Base64 encoded representation of the data

track2

Contains the data read from track 2.

track3

Contains the data read from track 3.

chip

Contains the ATR data read from the chip. For contactless chip card readers, multiple identification information can be returned if the card reader detects more than one chip. Each chip identification information is returned as an individual *data* array element.

security

Contains the data returned by the security module.

security/data

The security data can be one of the following:

- readLevel1 - The security data readability level is 1.
- readLevel2 - The security data readability level is 2.
- readLevel3 - The security data readability level is 3.
- readLevel4 - The security data readability level is 4.
- readLevel5 - The security data readability level is 5.
- badReadLevel - The security data reading quality is not acceptable.
- noData - There are no security data on the card.
- dataInvalid - The validation of the security data with the specific data on the magnetic stripe was not successful.
- hardwareError - The security module could not be used because of a hardware error.
- notInitialized - The security module could not be used because it was not initialized (e.g. CIM key is not loaded).

watermark

Contains the data read from the Swedish Watermark track.

memoryChip

Memory Card Identification data read from the memory chip.

memoryChip/data

The memory card protocol used to communicate with the card followed by the data. The memory card protocol can be one of the following:

- chipT0 - The card reader can handle the T=0 protocol.
- chipT1 - The card reader can handle the T=0 protocol.
- chipProtocolNotRequired - The carder is capable of communicating with the chip without requiring the application to specify any protocol.
- chipTypeAPart3 - The card reader can handle the ISO 14443 (Part3) Type A contactless chip card protocol.
- chipTypeAPart4 - The card reader can handle the ISO 14443 (Part4) Type A contactless chip card protocol.
- chipTypeB - The card reader can handle the ISO 14443 Type B contactless chip card protocol.
- chipTypeNFC - The card reader can handle the ISO 18092 (106/212/424kbps) contactless chip card protocol.

track1Front

Contains the data read from the front track 1. In some countries this track is known as JIS II track.

frontImage

Contains the full path and file name of the BMP image file for the front of the card.

backImage

Contains the the full path and file name of the BMP image file for the back of the card.

track1JIS

Contains the data read from JIS I track 1 (8bits/char).

track3JIS

data read from JIS I track 3 (8bits/char).

ddi

Contains the dynamic digital identification data read from magnetic stripe.

Event Messages

- [CardReader.InsertCardEvent](#)
- [CardReader.MediaInsertedEvent](#)
- [CardReader.MediaRemovedEvent](#)
- [CardReader.InvalidMediaEvent](#)
- [CardReader.TrackDetectedEvent](#)

3.3.8 - CardReader.WriteRawData

For motor-driven card readers, the ID card unit checks whether a card has been inserted. If so, the data is written to the tracks.

If no card has been inserted, and for all other categories of devices, the ID card unit waits for the application specified [timeout](#) for a card to be either inserted or pulled through. The next step is writing the data to the respective tracks.

The [CardReader.InsertCardEvent](#) event will be generated when there is no card in the card reader and the device is ready to accept a card.

The application must pass the magnetic stripe data in ASCII without any sentinels. The data will be converted by the Service Provider (ref [CardReader.ReadRawData](#)). If the data passed in is too long the invalidError error code will be returned.

This procedure is followed by data verification.

If power fails during a write the outcome of the operation will be vendor specific, there is no guarantee that the write will have succeeded.

Command Message

| Payload | Type | Required |
|---|----------------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "data": [{ | array (object) | |
| "destination": "track1", | string | |
| "data": "QmFzZTYOIGVuY29kZWQgZGF0YQ==", | string | |
| "writeMethod": "auto" | string | |
| }] | | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

data

An array of card data structures

data/

A card data structure

data/destination

Specifies where the card data is to be written to as one of the following:

- track1 - data is to be written to track 1.
- track2 - data is to be written to track 2.
- track3 - data is to be written to track 3.
- track1Front - data is to be written to the front track 1. In some countries this track is known as JIS II track.
- track1JIS - data is to be written to JIS I track 1 (8bits/char).
- track3JIS - data is to be written to JIS I track 3 (8bits/char).

data/data

Base4 encoded representation of the data

data/writeMethod

Indicates whether a low coercivity or high coercivity magnetic stripe is to be written as one of the following:

- loco - Write using low coercivity.
- hico - Write using high coercivity.
- auto - Service Provider will determine whether low or high coercivity is to be used.

default: "auto"

Completion Message

This event notifies the completion of the command and if successful includes the requested data.

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "mediaJam" | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- mediaJam - The card is jammed. Operator intervention is required.
- shutterFail - The open of the shutter failed due to manipulation or hardware error. Operator intervention is required.
- noMedia - The card was removed before completion of the write action (the event [CardReader.MediaInsertedEvent](#) has been generated). For motor driven devices, the write is disabled; i.e. another command has to be issued to enable the reader for card entry.
- invalidMedia - No track found; card may have been inserted or pulled through the wrong way.

- `writeMethod` - The `writeMethod` value is inconsistent with device capabilities.
- `cardTooShort` - The card that was inserted is too short. When this error occurs the card remains at the exit slot.
- `cardTooLong` - The card that was inserted is too long. When this error occurs the card remains at the exit slot.

Event Messages

- `CardReader.InsertCardEvent`
- `CardReader.MediaInsertedEvent`
- `CardReader.MediaRemovedEvent`
- `CardReader.InvalidTrackDataEvent`
- `CardReader.InvalidMediaEvent`

3.3.9 - CardReader.ChipIO

This command is used to communicate with the chip. Transparent data is sent from the application to the chip and the response of the chip is returned transparently to the application.

The identification information e.g. ATR of the chip must be obtained before issuing this command. The identification information for a user card or the Memory Card Identification (when available) must initially be obtained using `CardReader.ReadRawData`. The identification information for subsequent resets of a user card can be obtained using either `CardReader.ReadRawData` `CardReader.ChipPower`. The ATR for permanent connected chips is always obtained through `CardReader.ChipPower`.

For contactless chip card readers, applications need to specify which chip to contact with, as part of `chipData`, if more than one chip has been detected and multiple identification data has been returned by the `CardReader.ReadRawData` command.

For contactless chip card readers a collision of two or more card signals may happen. In this case, if the device is not able to pick the strongest signal, the `cardCollision` error code will be returned.

Command Message

| Payload | Type | Required |
|--------------------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "chipProtocol": Add example to YAML, | string | |

```
"chipData": Add example to YAML      string
}
```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

chipProtocol

Identifies the protocol that is used to communicate with the chip. Possible values are those described in CardReader.Capabilities. This field is ignored in communications with Memory Cards. The Service Provider knows which memory card type is currently inserted and therefore there is no need for the application to manage this.

chipData

The Base64 encoded data to be sent to the chip.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "mediaJam", | string | |
| "chipProtocol": Add example to YAML, | string | |
| "chipData": Add example to YAML | string | |
| } | | |

Properties

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- mediaJam - The card is jammed. Operator intervention is required.
- noMedia - There is no card inside the device.
- invalidMedia - No chip found; card may have been inserted the wrong way.
- invalidData - An error occurred while communicating with the chip.
- protocolNotSupported - The protocol used was not supported by the Service Provider.
- atrNotObtained - The ATR has not been obtained.
- cardCollision - There was an unresolved collision of two or more contactless card signals.

chipProtocol

Identifies the protocol that is used to communicate with the chip. This field contains the same value as the corresponding field in the payload. This field should be ignored in Memory Card dialogs and will contain *notSupported* when returned for any Memory Card dialog.

chipData

The Base64 encoded data received from the chip.

Event Messages

- [CardReader.MediaRemovedEvent](#)

3.3.10 - CardReader.Reset

This command is used by the application to perform a hardware reset which will attempt to return the card reader device to a known good state. This command does not over-ride a lock obtained by another application or service handle.

If the device is a user ID card unit, the device will attempt to either retain, eject or will perform no action on any user cards found in the device as specified in the input parameter. It may not always be possible to retain or eject the items as specified because of hardware problems. If a user card is found inside the device the

[CardReader.MediaInsertedEvent](#) will inform the application where card was actually moved to. If no action is specified the user card will not be moved even if this means that the device cannot be recovered.

If the device is a permanent chip card unit, this command will power-off the chip.

For devices with parking station capability there will be one *MediaInsertedEvent* for each card found.

Command Message

| Payload | Type | Required |
|--|------|----------|
| { "timeout": 5000, integer "resetIn": "retain" string } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

resetIn

Specifies the action to be performed on any user card found within the ID card unit as one of the following:

- eject - Eject any card found.

- retain - Retain any card found.
- noAction - No action should be performed on any card found.

Completion Message

This event notifies the completion of the command and if successful includes the requested data.

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "mediaJam" | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- mediaJam - The card is jammed. Operator intervention is required.
- shutterFail - The device is unable to open and close its shutter.
- retainBinFull - The retain bin is full; no more cards can be retained. The current card is still in the device.

Event Messages

- CardReader.MediaRemovedEvent

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

- [CardReader.RetainBinThresholdEvent](#)
- [CardReader.MediaDetectedEvent](#)

[3.3.11 - CardReader.ChipPower](#)

This command handles the power actions that can be done on the chip.

For user chips, this command is only used after the chip has been contacted for the first time using the [CardReader.ReadRawData](#) command. For contactless user chips, this command may be used to deactivate the contactless card communication.

For permanently connected chip cards, this command is the only way to control the chip power.

Command Message

| Payload | Type | Required |
|---------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "chipPower": "cold" | string | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

chipPower

Specifies the action to perform as one of the following:

- cold - The chip is powered on and reset.
- warm - The chip is reset.
- off - The chip is powered off.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "chipPowerNotSupported", | string | |
| "chipData": Add example to YAML | string | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- chipPowerNotSupported - The specified action is not supported by the hardware device.
- mediaJam - The card is jammed (only applies to contact user chips). Operator intervention is required.
- noMedia - There is no card inside the device (may not apply for contactless user chips).
- invalidMedia - No chip found; card may have been inserted or pulled through the wrong way.
- invalidData - An error occurred while communicating with the chip.
- atrNotObtained - The ATR has not been obtained (only applies to user chips).

chipData

The Base64 encoded data received from the chip.

Event Messages

- [CardReader.MediaRemovedEvent](#)

[3.3.12 - CardReader.ParkCard](#)

This command is used to move a card that is present in the reader to a parking station. A parking station is defined as an area in the ID card unit, which can be used to temporarily store the card while the device performs operations on another card. This command is also used to move a card from the parking station to the read/write, chip I/O or transport position. When a card is moved from the parking station to the read/write, chip I/O or transport position (*parkOut*), the read/write, chip I/O or transport position must not be occupied with another card, otherwise the error *cardPresent* will be returned.

After moving a card to a parking station, another card can be inserted and read by calling, e.g., [CardReader.ReadRawData](#) or [CardReader.ReadTrack](#).

Cards in parking stations will not be affected by any CardReader commands until they are removed from the parking station using this command, except for the [CardReader.Reset](#) command, which will move the cards in the parking stations as specified in its input as part of the reset action if possible.

Command Message

| Payload | Type | Required |
|--|------------------------------|----------|
| { "timeout": 5000, "direction": "out", "parkingStation": 0 } | integer string integer | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

direction

Specifies which way to move the card as one of the following values:

- in - The card is moved to the parking station from the read/write, chip I/O or transport position.
- out - The card is moved from the parking station to the read/write, chip I/O or transport position. Once the card has been moved any command can be executed e.g. [CardReader.EjectCard](#) or [CardReader.ReadRawData](#).

parkingStation

Specifies which parking station should be used for this command. This value is the same index as is identified in the [parkingStationMedia](#) array of [Common.Status](#).

Completion Message

This event notifies the completion of the command and if successful includes the requested data.

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "mediaJam" | string | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- mediaJam - The card is jammed. Operator intervention is required.
- noMedia - No card is present at the read/write, chip I/O or transport position and the *in* option has been selected. Or no card is in the parking station and the *out* option has been selected.
- cardPresent - Another card is present and is preventing successful movement of the card specified by [parkingStation](#).
- positionInvalid - The specified parking station is invalid.

Event Messages

None

[3.3.13 - CardReader.EMVClessConfigure](#)

This command is used to configure an intelligent contactless card reader before performing a contactless transaction. This command sets terminal related data elements, the list of terminal acceptable applications with associated application specific data and any encryption key data required for offline data authentication.

This command should be used prior to [CardReader.EMVClessPerformTransaction](#) if the command. It may be called once on application start up or when any of the configuration parameters require to be changed. The configuration set by this command is persistent.

This command should be called with a complete list of acceptable payment system applications as any previous configurations will be replaced.

Command Message

| Payload | Type | Required |
|--------------------------------------|----------------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "terminalData": Add example to YAML, | string | |
| "aidData": [{ | array (object) | |
| "aid": Add example to YAML, | string | |

```

"partialSelection": false,           boolean
"transactionType": 0,              integer
"kernelIdentifier": Add example to YAML, string
"configData": Add example to YAML    string
},
"keyData": [{                      array (object)
  "rid": Add example to YAML,       string
  "caPublicKey": {                  object
    "index": 0,                   integer
    "algorithmIndicator": 0,      integer
    "exponent": Add example to YAML, string
    "modulus": Add example to YAML, string
    "checksum": Add example to YAML string
  }
}
}

```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

terminalData

Base64 encoded representation of the BER-TLV formatted data for the terminal e.g. Terminal Type, Transaction Category Code, Merchant Name & Location etc. Any terminal based data elements referenced in the Payment Systems Specifications or EMVCo Contactless Payment Systems Specifications Books may be included (see References [2] to

[14] section for more details).

aidData

Specifies the list of acceptable payment system applications. For EMVCo approved contactless card readers each AID is associated with a Kernel Identifier and a Transaction Type. Legacy approved contactless readers may use only the AID.

Each AID-Transaction Type or each AID-Kernel-Transaction Type combination will have its own unique set of configuration data. See References [2] and [3] for more details.

aidData/aid

The application identifier to be accepted by the contactless chip card reader. The [CardReader.EMVClessQueryApplications](#) command will return the list of supported application identifiers.

aidData/partialSelection

If *partialSelection* is *true*, partial name selection of the specified AID is enabled. If *partialSelection* is *false*, partial name selection is disabled. A detailed explanation for partial name selection is given in EMV 4.3 Book 1, Section 11.3.5.

aidData/transactionType

The transaction type supported by the AID. This indicates the type of financial transaction represented by the first two digits of the ISO 8583:1987 Processing Code.

aidData/kernelIdentifier

Base64 encoded representation of the EMVCo defined kernel identifier associated with the *aid*. This field will be ignored if the reader does not support kernel identifiers.

aidData/configData

Base64 encoded representation of the list of BER-TLV formatted configuration data, applicable to the specific AID-Kernel ID-Transaction Type combination. The appropriate payment systems specifications define the BER-TLV tags to be configured.

keyData

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Specifies the encryption key information required by an intelligent contactless chip card reader for offline data authentication.

keyData/rid

Specifies the payment system's Registered Identifier (RID). RID is the first 5 bytes of the AID and identifies the payments system.

keyData/caPublicKey

CA Public Key information for the specified *rid*

keyData/caPublicKey/index

Specifies the CA Public Key Index for the specific RID.

keyData/caPublicKey/algorithmIndicator

Specifies the algorithm used in the calculation of the CA Public Key checksum. A detailed description of secure hash algorithm values is given in EMV Book 2, Annex B3; see reference [2]. For example, if the EMV specification indicates the algorithm is '01', the value of the algorithm is coded as 0x01.

keyData/caPublicKey/exponent

Base64 encoded representation of the CA Public Key Exponent for the specific RID. This value is represented by the minimum number of bytes required. A detailed description of public key exponent values is given in EMV Book 2, Annex B2; see reference [2]. For example, representing value '2¹⁶ + 1' requires 3 bytes in hexadecimal (0x01, 0x00, 0x01), while value '3' is coded as 0x03.

keyData/caPublicKey/modulus

Base64 encoded representation of the CA Public Key Modulus for the specific RID.

keyData/caPublicKey/checksum

Base64 encoded representation of the 20 byte checksum value for the CA Public Key.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "invalidTerminalData" | string | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- invalidTerminalData - Input data **terminalData** was invalid. Contactless chip card reader could not be configured successfully.
- invalidAidData - Input data **aidData** was invalid. Contactless chip card reader could not be configured successfully.
- invalidKeyData - Input data **keyData** was invalid. Contactless chip card reader could not be configured successfully.

Event Messages

None

3.3.14 - CardReader.EMVClassPerformTransaction

This command is used to enable an intelligent contactless card reader. The transaction will start as soon as the card tap is detected.

Based on the configuration of the contactless chip card and the reader device, this command could return data formatted either as magnetic stripe information or as a set of BER-TLV encoded EMV tags.

This command supports magnetic stripe emulation cards and EMV-like contactless cards but cannot be used on storage contactless cards. The latter must be managed using the [CardReader.ReadRawData](#) and [CardReader.ChipIO](#) commands.

For specific payment system's card profiles an intelligent card reader could return a set of EMV tags along with magnetic stripe formatted data. In this case, two contactless card data structures will be returned, one containing the magnetic stripe like data and one containing BER-TLV encoded tags.

If no card has been tapped, the contactless chip card reader waits for the period of time specified in the command call for a card to be tapped.

For intelligent contactless card readers, any in-built audio/visual feedback such as Beep/LEDs, need to be controlled directly by the reader. These indications should be implemented based on the EMVCo and payment system's specifications.

Command Message

| Payload | Type | Required |
|-----------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "data": Add example to YAML | string | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

data

Base64 encoded representation of the EMV data elements in a BER-TLV format required to perform a transaction. The types of object that could be included are:

- Transaction Type (9C)
- Amount Authorized (9F02)
- Transaction Date (9A)*
- Transaction Time (9F21)*
- Transaction Currency Code (5F2A)

Individual payment systems could define further data elements.

Tags are not mandatory with this command and this value can be omitted.

*Tags 9A and 9F21 could be managed internally by the reader. If tags are not supplied, tag values may be used from the configuration sent previously in the [CardReader.EMVClessConfigure](#) command.

Completion Message

| Payload | Type | Required |
|--|-------------|-----------------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "noMedia", | string | |
| "track1": { | object | |
| "txOutcome": "multipleCards", | string | |
| "cardholderAction": "none", | string | |
| "dataRead": Add example to YAML, | string | |
| "clessOutcome": { | object | |
| "cvm": "onlinePIN", | string | |
| "alternateInterface": "contact", | string | |
| "receipt": false, | boolean | |
| "uiOutcome": { | object | |
| "messageId": 0, | integer | |

```

"status": "notReady",           string
"holdTime": 0,                  integer
"valueQualifier": "amount",    string
"value": "123.45",             string
"currencyCode": "GBP",          string
"languagePreferenceData": "en"  string
},
"uiRestart": {

```

See [uiOutcome](#) properties.

```

},
"fieldOffHoldTime": 0,           integer
"cardRemovalTimeout": 0,         integer
"discretionaryData": Add example to YAML string
}
},
"track2": {

```

See [track1](#) properties.

```

},
"track3": {

```

See [track1](#) properties.

```

},
"chip": {

```

See [track1](#) properties.

```

}
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- noMedia - The card was removed before completion of the read operation.
- invalidMedia - No track or chip was found or the card tapped cannot be used with this command (e.g. contactless storage cards).
- readerNotConfigured - This command was issued before calling [CardReader.EMVClassConfigure](#) command.

track1

Contains the chip returned data formatted in as track 1. This value is set after the contactless transaction has been completed with mag-stripe mode.

track1/txOutcome

If multiple data sources are returned, this parameter should be the same for each one. Specifies the contactless transaction outcome as one of the following flags:

- multipleCards - Transaction could not be completed as more than one contactless card was tapped.
- approve - Transaction was approved offline.
- decline - Transaction was declined offline.
- onlineRequest - Transaction was requested for online authorization.
- onlineRequestCompletionRequired - Transaction requested online authorization and will be completed after a re-tap of the card. Transaction should be completed by issuing the [CardReader.EMVClassIssuerUpdate](#) command.
- tryAgain - Transaction could not be completed due to a card read error. The contactless card could be tapped again to re-attempt the transaction.
- tryAnotherInterface - Transaction could not be completed over the contactless interface. Another interface may be suitable for this transaction (for example contact).

- endApplication - Transaction cannot be completed on the contactless card due to an irrecoverable error.
- confirmationRequired - Transaction was not completed as a result of a requirement to allow entry of confirmation code on a mobile device. Transaction should be completed by issuing the [CardReader.EMVClassPerformTransaction](#) after a card removal and a re-tap of the card.

NOTE: The values for outcome have been mapped against the EMV Entry Point Outcome structure values defined in the EMVCo Specifications for Contactless Payment Systems (Book A and B).

track1/cardholderAction

Specifies the cardholder action as one of the following:

- none - Transaction was completed. No further action is required.
- retap - The contactless card should be re-tapped to complete the transaction. This value can be returned when [txOutcome](#) is *onlineRequestCompletionRequired* or *confirmationRequired*.
- holdCard - The contactless card should not be removed from the field until the transaction is completed.

track1/dataRead

The Base64 encoded representation of the data read from the chip after a contactless transaction has been completed successfully. If the member name is [chip](#), the BER-TLV formatted data contains cryptogram tag (9F26) after a contactless chip transaction has been completed successfully. If the member name is [track1](#), [track2](#) or [track3](#) this contains the data read from the chip, i.e the value returned by the card reader device and no cryptogram tag (9F26). This value is terminated with a single null character and cannot contain UNICODE characters.

track1/clessOutcome

The Entry Point Outcome specified in EMVCo Specifications for Contactless Payment Systems (Book A and B). This can be omitted for contactless chip card readers that do not follow EMVCo Entry Point Specifications.

track1/clessOutcome/cvm

Specifies the cardholder verification method (CVM) to be performed as one of the following:

- onlinePIN - Online PIN should be entered by the cardholder.

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

- confirmationCodeVerified - A confirmation code entry has been successfully done on a mobile device.
- sign - Application should obtain cardholder signature.
- noCVM - No CVM is required for this transaction.
- noCVMPreference - There is no CVM preference, but application can follow the payment system's rules to process the transaction.

track1/clessOutcome/alternateInterface

If **txOutcome** is not *tryAnotherInterface*, this should be ignored. If **txOutcome** is *tryAnotherInterface*, this specifies the alternative interface to be used to complete a transaction as one of the following:

- contact - Contact chip interface should be used to complete a transaction.
- magneticStripe - Magnetic stripe interface should be used to complete a transaction.

track1/clessOutcome/receipt

Specifies whether a receipt should be printed. True indicates that a receipt is required.

track1/clessOutcome/uiOutcome

The user interface details required to be displayed to the cardholder after processing the outcome of a contactless transaction. If no user interface details are required, this will be omitted. Please refer to EMVCo Contactless Specifications for Payment Systems Book A, Section 6.2 for details of the data within this object.

track1/clessOutcome/uiOutcome/messageId

Represents the EMVCo defined message identifier that indicates the text string to be displayed, e.g., 0x1B is the “Authorising Please Wait” message (see EMVCo Contactless Specifications for Payment Systems Book A, Section 9.4).

track1/clessOutcome/uiOutcome/status

Represents the EMVCo defined transaction status value to be indicated through the Beep/LEDs as one of the following:

- notReady - Contactless card reader is not able to communicate with a card. This status occurs towards the end of a contactless transaction or if the reader is not powered on.
- idle - Contactless card reader is powered on, but the reader field is not yet active for communication with a card.

- readyToRead - Contactless card reader is powered on and attempting to initiate communication with a card.
- processing - Contactless card reader is in the process of reading the card.
- cardReadOk - Contactless card reader was able to read a card successfully.
- processingError - Contactless card reader was not able to process the card successfully.

track1/clessOutcome/uiOutcome/holdTime

Represents the hold time in units of 100 milliseconds for which the application should display the message before processing the next user interface data.

track1/clessOutcome/uiOutcome/valueQualifier

Qualifies [value](#). This data is defined by EMVCo as either “Amount” or “Balance” as one of the following:

- amount - *value* is an Amount.
- balance - *value* is a Balance.
- notApplicable - *value* is neither of the above.

default: notApplicable

track1/clessOutcome/uiOutcome/value

Represents the value of the amount or balance (as specified by [valueQualifier](#)) to be displayed where appropriate.

track1/clessOutcome/uiOutcome/currencyCode

Represents the numeric value of currency code as per ISO 4217.

default:

track1/clessOutcome/uiOutcome/languagePreferenceData

Represents the language preference (EMV Tag ‘5F2D’) if returned by the card. The application should use this data to display all messages in the specified language until the transaction concludes.

default:

track1/clessOutcome/uiRestart

The user interface details required to be displayed to the cardholder when a transaction needs to be completed with a re-tap. If no user interface details are required, this will be omitted.

track1/clessOutcome/fieldOffHoldTime

The application should wait for this specific hold time in units of 100 milliseconds, before re-enabling the contactless card reader by issuing either the [CardReader.EMVClessPerformTransaction](#) command or the [CardReader.EMVClessIssuerUpdate](#) command depending on the value of [txOutcome](#). For intelligent contactless card readers, the completion of this command ensures that the contactless chip card reader field is automatically turned off, so there is no need for the application to disable the field.

track1/clessOutcome/cardRemovalTimeout

Specifies a timeout value in units of 100 milliseconds for prompting the user to remove the card.

track1/clessOutcome/discretionaryData

Base64 encoded representation of the payment system's specific discretionary data read from the chip, in a BER-TLV format, after a contactless transaction has been completed. If discretionary data is not present, this will be omitted.

track2

Contains the chip returned data formatted in as track 2. This value is set after the contactless transaction has been completed with mag-stripe mode.

track3

Contains the chip returned data formatted in as track 3. This value is set after the contactless transaction has been completed with mag-stripe mode.

chip

Contains the BER-TLV formatted data read from the chip. This value is set after the contactless transaction has been completed with EMV mode or mag-stripe mode.

Event Messages

- [CardReader.EMVClessReadStatusEvent](#)
- [CardReader.MediaRemovedEvent](#)

3.3.15 - CardReader.EMVClessIssuerUpdate

This command performs the post authorization processing on payment systems contactless cards.

Before an online authorized transaction is considered complete, further chip processing may be requested by the issuer. This is only required when the authorization response includes issuer update data; either issuer scripts or issuer authentication data.

The command enables the contactless card reader and waits for the customer to re-tap their card.

The contactless chip card reader waits for the period of time specified in the WFSEExecute call for a card to be tapped.

Command Message

| Payload | Type | Required |
|-----------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "data": Add example to YAML | string | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

data

Base64 encoded representation of the EMV data elements in a BER-TLV format received

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

from the authorization response that are required to complete the transaction processing.
The types of object that could be listed in lpData are:

- Authorization Code (if present)
- Issuer Authentication Data (if present)
- Issuer Scripts or proprietary payment system's data elements (if present) and any other data elements if required.

Completion Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "noMedia", | string | |
| "chip": { | object | |
| "txOutcome": "multipleCards", | string | |
| "cardholderAction": "none", | string | |
| "dataRead": Add example to YAML, | string | |
| "clssOutcome": { | object | |
| "cvm": "onlinePIN", | string | |
| "alternateInterface": "contact", | string | |
| "receipt": false, | boolean | |
| "uiOutcome": { | object | |
| "messageId": 0, | integer | |
| "status": "notReady", | string | |
| "holdTime": 0, | integer | |
| "valueQualifier": "amount", | string | |
| "value": "123.45", | string | |
| "currencyCode": "GBP", | string | |
| "languagePreferenceData": "en" | string | |

```

},
"uiRestart": {          object
  See uiOutcome properties.
},
"fieldOffHoldTime": 0,    integer
"cardRemovalTimeout": 0,   integer
"discretionaryData": Add example to YAML  string
}
}
}

```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- noMedia - The card was removed before completion of the read action.
- invalidMedia - No track or chip found or card tapped cannot be used with this command (e.g. contactless storage cards or a different card than what was used to complete the [CardReader.EMVClessPerformTransaction](#) command).
- transactionNotInitiated - This command was issued before calling the [CardReader.EMVClessPerformTransaction](#) command.

chip

Contains the BER-TLV formatted data read from the chip. This will be omitted if no data has

been returned.

chip/txOutcome

If multiple data sources are returned, this parameter should be the same for each one. Specifies the contactless transaction outcome as one of the following flags:

- multipleCards - Transaction could not be completed as more than one contactless card was tapped.
- approve - Transaction was approved offline.
- decline - Transaction was declined offline.
- onlineRequest - Transaction was requested for online authorization.
- onlineRequestCompletionRequired - Transaction requested online authorization and will be completed after a re-tap of the card. Transaction should be completed by issuing the [CardReader.EMVClassIssuerUpdate](#) command.
- tryAgain - Transaction could not be completed due to a card read error. The contactless card could be tapped again to re-attempt the transaction.
- tryAnotherInterface - Transaction could not be completed over the contactless interface. Another interface may be suitable for this transaction (for example contact).
- endApplication - Transaction cannot be completed on the contactless card due to an irrecoverable error.
- confirmationRequired - Transaction was not completed as a result of a requirement to allow entry of confirmation code on a mobile device. Transaction should be completed by issuing the [CardReader.EMVClassPerformTransaction](#) after a card removal and a re-tap of the card.

NOTE: The values for outcome have been mapped against the EMV Entry Point Outcome structure values defined in the EMVCo Specifications for Contactless Payment Systems (Book A and B).

chip/cardholderAction

Specifies the cardholder action as one of the following:

- none - Transaction was completed. No further action is required.
- retap - The contactless card should be re-tapped to complete the transaction. This value can be returned when `txOutcome` is `onlineRequestCompletionRequired` or `confirmationRequired`.
- holdCard - The contactless card should not be removed from the field until the transaction is completed.

chip/dataRead

The Base64 encoded representation of the data read from the chip after a contactless transaction has been completed successfully. If the member name is `chip`, the BER-TLV formatted data contains cryptogram tag (9F26) after a contactless chip transaction has been completed successfully. If the member name is `track1`, `track2` or `track3` this contains the data read from the chip, i.e the value returned by the card reader device and no cryptogram tag (9F26). This value is terminated with a single null character and cannot contain UNICODE characters.

chip/clessOutcome

The Entry Point Outcome specified in EMVCo Specifications for Contactless Payment Systems (Book A and B). This can be omitted for contactless chip card readers that do not follow EMVCo Entry Point Specifications.

chip/clessOutcome/cvm

Specifies the cardholder verification method (CVM) to be performed as one of the following:

- `onlinePIN` - Online PIN should be entered by the cardholder.
- `confirmationCodeVerified` - A confirmation code entry has been successfully done on a mobile device.
- `sign` - Application should obtain cardholder signature.
- `noCVM` - No CVM is required for this transaction.
- `noCVMPreference` - There is no CVM preference, but application can follow the payment system's rules to process the transaction.

chip/clessOutcome/alternateInterface

If `txOutcome` is not `tryAnotherInterface`, this should be ignored. If `txOutcome` is `tryAnotherInterface`, this specifies the alternative interface to be used to complete a transaction as one of the following:

- `contact` - Contact chip interface should be used to complete a transaction.
- `magneticStripe` - Magnetic stripe interface should be used to complete a transaction.

chip/clessOutcome/receipt

Specifies whether a receipt should be printed. True indicates that a receipt is required.

chip/clessOutcome/uiOutcome

The user interface details required to be displayed to the cardholder after processing the

outcome of a contactless transaction. If no user interface details are required, this will be omitted. Please refer to EMVCo Contactless Specifications for Payment Systems Book A, Section 6.2 for details of the data within this object.

chip/clessOutcome/uiOutcome/messageId

Represents the EMVCo defined message identifier that indicates the text string to be displayed, e.g., 0x1B is the “Authorising Please Wait” message (see EMVCo Contactless Specifications for Payment Systems Book A, Section 9.4).

chip/clessOutcome/uiOutcome/status

Represents the EMVCo defined transaction status value to be indicated through the Beep/LEDs as one of the following:

- notReady - Contactless card reader is not able to communicate with a card. This status occurs towards the end of a contactless transaction or if the reader is not powered on.
- idle - Contactless card reader is powered on, but the reader field is not yet active for communication with a card.
- readyToRead - Contactless card reader is powered on and attempting to initiate communication with a card.
- processing - Contactless card reader is in the process of reading the card.
- cardReadOk - Contactless card reader was able to read a card successfully.
- processingError - Contactless card reader was not able to process the card successfully.

chip/clessOutcome/uiOutcome/holdTime

Represents the hold time in units of 100 milliseconds for which the application should display the message before processing the next user interface data.

chip/clessOutcome/uiOutcome/valueQualifier

Qualifies [value](#). This data is defined by EMVCo as either “Amount” or “Balance” as one of the following:

- amount - *value* is an Amount.
- balance - *value* is a Balance.
- notApplicable - *value* is neither of the above.

default: notApplicable

chip/clessOutcome/uiOutcome/value

Represents the value of the amount or balance (as specified by [valueQualifier](#)) to be displayed where appropriate.

chip/clessOutcome/uiOutcome/currencyCode

Represents the numeric value of currency code as per ISO 4217.

default:

chip/clessOutcome/uiOutcome/languagePreferenceData

Represents the language preference (EMV Tag ‘5F2D’) if returned by the card. The application should use this data to display all messages in the specified language until the transaction concludes.

default:

chip/clessOutcome/uiRestart

The user interface details required to be displayed to the cardholder when a transaction needs to be completed with a re-tap. If no user interface details are required, this will be omitted.

chip/clessOutcome/fieldOffHoldTime

The application should wait for this specific hold time in units of 100 milliseconds, before re-enabling the contactless card reader by issuing either the [CardReader.EMVClessPerformTransaction](#) command or the [CardReader.EMVClessIssuerUpdate](#) command depending on the value of [txOutcome](#). For intelligent contactless card readers, the completion of this command ensures that the contactless chip card reader field is automatically turned off, so there is no need for the application to disable the field.

chip/clessOutcome/cardRemovalTimeout

Specifies a timeout value in units of 100 milliseconds for prompting the user to remove the card.

chip/clessOutcome/discretionaryData

Base64 encoded representation of the payment system's specific discretionary data read from the chip, in a BER-TLV format, after a contactless transaction has been completed. If discretionary data is not present, this will be omitted.

Event Messages

- [CardReader.EMVClessReadStatusEvent](#)
- [CardReader.MediaRemovedEvent](#)

3.4 - Event Messages

[3.4.1 - CardReader.MediaRetainedEvent](#)

This specifies that the card was retained.

[3.4.2 - CardReader.InsertCardEvent](#)

This event notifies the application when the device is ready for the user to insert a card.

[3.4.3 - CardReader.MedialnsertedEvent](#)

This event notifies the application when the device is ready for the user to insert a card.

[3.4.4 - CardReader.InvalidMediaEvent](#)

This event specifies that the media the user is attempting to insert is not a valid card or it is a card but it is in the wrong orientation.

[3.4.5 - CardReader.TrackDetectedEvent](#)

This execute event notifies the application what track data the inserted card has, before the reading of the data has completed. This event will be posted once when tracks are detected during card insertion.

| Payload | Type | Required |
|---------|------|----------|
| { | | |

```
"track1": false,      boolean  
"track2": false,      boolean  
"track3": false,      boolean  
"watermark": false,   boolean  
"frontTrack1": false  boolean  
}  
  
Properties
```

track1

The card reader has track 1.

default: false

track2

The card reader has track 2.

default: false

track3

The card reader has track 3.

default: false

watermark

The card reader has the Swedish watermark track.

default: false

frontTrack1

The card reader has front track 1.

default: false

3.4.6 - CardReader.InvalidTrackDataEvent

This execute event specifies that a track contained invalid or no data.

| Payload | Type | Required |
|-------------------------------|--------|----------|
| { | | |
| "status": "missing", | string | |
| "track": Add example to YAML, | string | |
| "data": Add example to YAML | string | |
| } | | |

Properties

status

Status of reading the track as one of the following:

- missing - The track is blank.
- invalid - The data contained on the track is invalid.
- tooLong - The data contained on the track is too long.
- tooShort - The data contained on the track is too short.

track

The keyword of the track on which the error occurred.

data

Any data from the track that could be read.

3.4.7 - CardReader.MediaDetectedEvent

This is generated if media is detected during a [CardReader.Reset](#). The parameter on the event informs the application of the position of the card on the completion of the reset. For devices with parking station capability there will be one event for each card found.

| Payload | Type | Required |
|---------|------|----------|
|---------|------|----------|

```
{
  "resetOut": "ejected"  string
}
```

Properties

resetOut

Specifies the action that was performed on any card found within the device as one of the following:

- ejected - The card was ejected.
- retained - The card was retained.
- readPosition - The card is in read position.
- jammed - The card is jammed in the device.

3.4.8 - CardReader.EMVClessReadStatusEvent

This notifies that the communication (i.e. the commands exchanged linked to the tap) between the card and the intelligent contactless card reader are complete. The application can use this event to display intermediate messages, progress of card read, audio signals or anything else that might be required. The intelligent contactless card reader will continue the processing and the result of the processing will be returned in the output of the [CardReader.EMVClessPerformTransaction](#) command.

| Payload | Type | Required |
|--------------------------------|---------|----------|
| { | | |
| "messageId": 0, | integer | |
| "status": "notReady", | string | |
| "holdTime": 0, | integer | |
| "valueQualifier": "amount", | string | |
| "value": "123.45", | string | |
| "currencyCode": "GBP", | string | |
| "languagePreferenceData": "en" | string | |
| } | | |

Properties

messageId

Represents the EMVCo defined message identifier that indicates the text string to be displayed, e.g., 0x1B is the “Authorising Please Wait” message (see EMVCo Contactless Specifications for Payment Systems Book A, Section 9.4).

status

Represents the EMVCo defined transaction status value to be indicated through the Beep/LEDs as one of the following:

- notReady - Contactless card reader is not able to communicate with a card. This status occurs towards the end of a contactless transaction or if the reader is not powered on.
- idle - Contactless card reader is powered on, but the reader field is not yet active for communication with a card.
- readyToRead - Contactless card reader is powered on and attempting to initiate communication with a card.
- processing - Contactless card reader is in the process of reading the card.
- cardReadOk - Contactless card reader was able to read a card successfully.
- processingError - Contactless card reader was not able to process the card successfully.

holdTime

Represents the hold time in units of 100 milliseconds for which the application should display the message before processing the next user interface data.

valueQualifier

Qualifies [value](#). This data is defined by EMVCo as either “Amount” or “Balance” as one of the following:

- amount - *value* is an Amount.
- balance - *value* is a Balance.
- notApplicable - *value* is neither of the above.

default: notApplicable

value

Represents the value of the amount or balance (as specified by [valueQualifier](#)) to be displayed where appropriate.

currencyCode

Represents the numeric value of currency code as per ISO 4217.

default:

languagePreferenceData

Represents the language preference (EMV Tag '5F2D') if returned by the card. The application should use this data to display all messages in the specified language until the transaction concludes.

default:

3.5 - Unsolicited Messages

3.5.1 - [CardReader.MediaRemovedEvent](#)

This event specifies that the inserted card was manually removed by the user during the processing of a read command, during the processing of a [CardReader.ChipIO](#), [CardReader.ChipPower](#) command, during or after a [CardReader.RetainCard](#), [CardReader.Reset](#) operation, after an [CardReader.EjectCard](#) operation or after the card is removed by the user in a latched dip card unit.

3.5.2 - [CardReader.CardActionEvent](#)

This service event specifies that a card has been retained or ejected by either the automatic power on or power off action of the device.

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "action": "retained" , | string | |
| "position": "entering" | string | |

{

Properties

action

Specifies which action has been performed with the card. Possible values are:

- retained - The card has been retained.
- ejected - The card has been ejected.
- readPosition - The card has been moved to the read position.

position

Position of card before being retained or ejected. Possible values are:

- unknown - The position of the card cannot be determined.
- present - The card was present in the reader.
- entering - The card was entering the reader.

[3.5.3 - CardReader.RetainBinThresholdEvent](#)

This specifies that the retain bin holding the retained cards has reached a threshold condition or the threshold condition is removed.

| Payload | Type | Required |
|---------------|--------|----------|
| { | | |
| "state": "ok" | string | |
| } | | |

Properties

state

Specifies the state of the ID card unit retain bin as one of the following:

- ok - The retain bin of the ID card unit was emptied.
- full - The retain bin of the ID card unit is full.
- high - The retain bin of the ID card unit is nearly full.

4 - Cash Management Interface

This chapter defines the Cash Management interface functionality and messages.

4.1 - Summary

This specification describes the functionality of an XFS4IoT compliant Cash Management interface. It defines the service-specific commands that can be issued to the service using the WebSocket endpoint.

This interface is to be used together with Dispenser and/or CashAcceptor interfaces to handle management of cash units, cash counts and banknote information.

4.2 - General Information

4.2.1 - Note Classification

Notes are classified by the XFS4IoT specification according to the following definitions:

1. Level 1 -- Note not recognized.
2. Level 2 -- Recognized counterfeit note.
3. Level 3 -- Suspected counterfeit note.
4. Level 4 -- Recognized note that is identified as genuine. This includes notes which are fit or unfit for recycling.

This definition allows support for legislative note handling standards that may exist in various countries and economic regions. Local requirements or device capability may dictate that notes are not classified as level 2 and level 3; the P6 string reported by Common.Capabilities *CashAcceptor.counterfeitAction* reports whether notes are classified into all 4 levels and whether level 2 or 3 notes can be returned to the customer.

The above classification levels can be used to support note handling functionality which includes:

1. The ability to remove counterfeit notes from circulation.
2. Reporting of recognized, counterfeit and suspected counterfeit notes.
3. Creating and reporting of note signatures in order to allow back-tracing of notes.

A note's classification can be changed based on the note's serial number, currency and value by specifying a classification list. A classification list can be used to re-classify a matching note to a lower level, including classifying a genuine note as unfit for dispensing. Once reclassified, the note will be automatically handled according to the local country specific note handling standard or legislation for the note's new note classification, including any level 2 or 3 note retention rules. Any reclassification will result in the normal events and behavior, for example a `CashManagement.InfoAvailable` event will reflect the note's reclassification. Reclassification can be used to make dynamic changes to note handling procedures without a software upgrade, enabling functionality such as taking older notes out of circulation or handling of counterfeit notes on a local basis.

Reclassification cannot be used to change a note's classification to a higher level, for example, a note recognized as counterfeit by the device cannot be reclassified as genuine. In addition, it is not possible to re-classify a level 2 note as level 1. No particular use case has been identified for reclassifying Level 3 and 4 notes as level 1, but there is no reason to restrict this reclassification.

Classification lists can be specified using `CashManagement.SetClassificationList` and retrieved using `CashManagement.GetClassificationList`. A classification list is a superset of the XFS 3.x blacklist; any items specified as level 2 in the classification list are considered part of the blacklist. However support for the old blacklist commands has been removed as it may lead to overlap and confusion.

The classification list functionality can use a mask to specify serial numbers. The mask is defined as follows: A '?' character (0x003F) is the wildcard used to match a single Unicode character, and a '*' character (0x002A) is the wildcard used to match one or more Unicode characters.

For example, "S8H9??16?4" would represent a match for the serial numbers "S8H9231654" and "S8H9761684". A mask of "HD90*2" would be used in order to match serial numbers that begin with "HD90" and end with "2", for example "HD9028882", "HD9083276112". Note that the mask can only use one asterisk, and if a real character is required then it must be preceded by a backslash, for example: '\\\' for a backslash, '*' for an asterisk or '\\?' for a question mark. Note that this flexibility means that it is possible to overlap definitions, for example "HD90*" and "HD902*" would both match on the serial number HD9028882".

4.3 - Command Messages

4.3.1 - `CashManagement.GetCashUnitInfo`

This command is used to obtain information regarding the status and contents of cash units.

It is possible that multiple cash units may be associated with one physical cash unit. This should only occur if the physical cash unit is capable of handling this situation, i.e. if it can

store multiple denominations and report meaningful count and replenishment information for each denomination or if it can store retracted and rejected items as separate units and report meaningful count and replenishment information for each of them.

Starting with XFS 4.0 the concept of logical cash units is removed from XFS services. Only physical units are reported. The former logical cash counts can be easily calculated from the given cash counts.

Command Message

| Payload | Type | Required |
|--------------------------------|---------|----------|
| { "timeout": 5000 } } | integer | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "cashunits": { | object | |
| "additionalProperties": { | object | |
| "status": "ok", | string | |
| "type": "billCassette", | string | |
| "currencyID": Add example to YAML, | string | |

| | |
|--|---------|
| "value": 0, | number |
| "logicalCount": 0, | integer |
| "maximum": 0, | integer |
| "appLock": false, | boolean |
| "cashUnitName": Add example to YAML, | string |
| "initialCount": 0, | integer |
| "dispensedCount": 0, | integer |
| "presentedCount": 0, | integer |
| "retractedCount": 0, | integer |
| "rejectCount": 0, | integer |
| "minimum": 0, | integer |
| "physicalPositionName": Add example to YAML, | string |
| "unitID": Add example to YAML, | string |
| "count": 0, | integer |
| "maximumCapacity": 0, | integer |
| "hardwareSensor": false, | boolean |
| "itemType": { | object |
| "all": false, | boolean |
| "unfit": false, | boolean |
| "individual": false, | boolean |
| "level1": false, | boolean |
| "level2": false, | boolean |
| "level3": false, | boolean |
| "itemProcessor": false, | boolean |
| "unfitIndividual": false | boolean |
| } | |

```

"cashInCount": 0,           integer
"noteNumberList": {          object
"noteNumber": [{             array (object)
"noteID": 0,               integer
"count": 0                 integer
}]
},
"noteIDs": [0]              array (integer)
}
}
}

```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

cashunits

Object containing cash unit information.

cashunits/additionalProperties

Cash unit information.

cashunits/additionalProperties/status

Supplies the status of the cash unit. Following values are possible:

- ok - The cash unit is in a good state.
- full - The cash unit is full.
- high - The cash unit is almost full (i.e. reached or exceeded the threshold defined by *maximum*).
- low - The cash unit is almost empty (i.e. reached or below the threshold defined by *minimum*).
- empty - The cash unit is empty, or insufficient items in the cash unit are preventing further dispense operations.
- inoperative - The cash unit is inoperative.
- missing - The cash unit is missing.
- noValue - The values of the specified cash unit are not available.
- noReference - There is no reference value available for the notes in this cash unit. The cash unit has not been calibrated.
- manipulated - The cash unit has been inserted (including removal followed by a reinsertion) when the device was not in the exchange state. This cash unit cannot be dispensed from.

`cashunits/additionalProperties/type`

Type of cash unit. Following values are possible:

- notApplicable - Not applicable. Typically means cash unit is missing.
- rejectCassette - Reject cash unit. This type will also indicate a combined reject/retract cash unit.
- billCassette - Cash unit containing bills.
- coinCylinder - Coin cylinder.
- coinDispenser - Coin dispenser as a whole unit.
- retractCassette - Retract cash unit.
- coupon - Cash unit containing coupons or advertising material.
- document - Cash unit containing documents.
- replenishmentContainer - Replenishment container. A cash unit can be refilled from a replenishment container.
- recycling - Recycling cash unit. This unit is only present when the device implements the Dispenser and CashAcceptor interfaces.
- cashIn - Cash-in cash unit.

`cashunits/additionalProperties/currencyID`

A three character string storing the ISO format [Ref. 2] Currency ID. This value will be omitted for cash units which contain items of more than one currency type or items to which currency is not applicable. If the *status* field for this cash unit is *noValue* it is the

responsibility of the client to assign a value to this field. This value is persistent.

cashunits/additionalProperties/value

Supplies the value of a single item in the cash unit. This value is expressed as floating point value. If the *currencyID* field for this cash unit is omitted, then this field will contain zero. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

cashunits/additionalProperties/logicalCount

The meaning of this count depends on the type of cash unit. This value is persistent. For all cash units except retract cash units (*type* is not *retractCassette*) this value specifies the number of items inside the cash unit. For all dispensing cash units (*type* is *billCassette*, *coinCylinder*, *coinDispenser*, *coupon*, *document* or *recycling*), this value includes any items from the cash unit not yet presented to the customer. This count is only decremented when the items are either known to be in customer access or successfully rejected. If the cash unit is usable from the CashAcceptor interface (*type* is *recycling*, *cashIn*, *retractCassette* or *rejectCassette*) then this value will be incremented as a result of a cash-in operation. Note that for a reject cash unit (*type* is *rejectCassette*), this value is unreliable, since the typical reason for dumping items to the reject cash unit is a suspected count failure. For a retract cash unit (*type* is *retractCassette*) this value specifies the number of retract operations which result in items entering the cash unit.

cashunits/additionalProperties/maximum

When *count* reaches this value the threshold event

CashManagement.CashUnitThresholdEvent (*high*) will be generated. This value can be different from the actual capacity of the cassette. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

cashunits/additionalProperties/appLock

If this value is TRUE items cannot be dispensed from or deposited into the cash unit. If this value is TRUE and the client attempts to use the cash unit a CashManagement.CashUnitErrorEvent event will be generated and an error completion message will be returned. This value is persistent.

cashunits/additionalProperties/cashUnitName

A name which helps to identify the type of the cash unit. This is especially useful in the case

of cash units of type *document* where different documents can have the same currency and value. For example, travelers checks and bank checks may have the same currency and value but still need to be identifiable as different types of document. Where this value is not relevant (e.g. in bill cash units) the property can be omitted. This value is persistent.

cashunits/additionalProperties/initialCount

Initial number of items contained in the cash unit. This value is persistent.

cashunits/additionalProperties/dispensedCount

The number of items dispensed from this cash unit. This count is incremented when the items are removed from the cash units. This count includes any items that were rejected during the dispense operation and are no longer in this cash unit. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

cashunits/additionalProperties/presentedCount

The number of items from this cash unit that have been presented to the customer. This count is incremented when the items are presented to the customer. If it is unknown if a customer has been presented with the items, then this count is not updated. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

cashunits/additionalProperties/retractedCount

The number of items that have been accessible to a customer and retracted into the cash unit. This value is persistent.

cashunits/additionalProperties/rejectCount

The number of items dispensed from this cash unit which have been rejected, are in a cash unit other than this cash unit, and which have not been accessible to a customer. This value may be unreliable, since a typical reason for rejecting items is a suspected pick failure. Other reasons for rejecting items may include incorrect note denominations, classifications not valid for dispensing, or where the transaction has been cancelled and a Reject command has been called. For reject and retract cash units (*type* is *rejectCassette* or *retractCassette*) this field does not apply and will be reported as zero. This value is persistent.

cashunits/additionalProperties/minimum

This field is not applicable to retract and reject cash units. For all cash units which dispense items (all other), when *count* reaches this value the threshold event

CashManagement.CashUnitThresholdEvent (*low*) will be generated. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

cashunits/additionalProperties/physicalPositionName

A name identifying the physical location of the cash unit.

cashunits/additionalProperties/unitID

A 5 character string uniquely identifying the cash unit.

cashunits/additionalProperties/count

As defined by the *logicalCount* description, but with the following exceptions: This count does not include items dispensed but not yet presented. On cash units with *type* set to "retractCassette" the count represents the number of items, unless the device cannot count items during a retract, in which case this count will be zero. This value is persistent.

cashunits/additionalProperties/maximumCapacity

The maximum number of items the cash unit can hold. This is only for informational purposes. No threshold event CashManagement.CashUnitThresholdEvent will be generated. This value is persistent.

cashunits/additionalProperties/hardwareSensor

Specifies whether or not threshold events can be generated based on hardware sensors in the device. If this value is TRUE then threshold events may be generated based on hardware sensors as opposed to counts.

cashunits/additionalProperties/itemType

Specifies the type of items the cash unit takes as a combination of the following flags. The table in the Comments section of this command defines how to interpret the combination of these flags (TODO: include Table)

cashunits/additionalProperties/itemType/all

The cash unit takes all fit banknote types. These are level 4 notes which are fit for recycling.

cashunits/additionalProperties/itemType/unfit

The cash unit takes all unfit banknotes. These are level 4 notes which are unfit for recycling.

cashunits/additionalProperties/itemType/individual

The cash unit takes all types of fit banknotes specified in an individual list. These are level 4 notes which are fit for recycling.

cashunits/additionalProperties/itemType/level1

Level 1 note types are stored in this cash unit.

cashunits/additionalProperties/itemType/level2

If notes can be classified as level 2, then level 2 note types are stored in this cash unit.

cashunits/additionalProperties/itemType/level3

If notes can be classified as level 3, then level 3 note types are stored in this cash unit.

cashunits/additionalProperties/itemType/itemProcessor

The cash unit can accept items on the ItemProcessor interface.

cashunits/additionalProperties/itemType/unfitIndividual

The cash unit takes all types of unfit banknotes specified in an individual list. These are level 4 notes which are unfit for recycling.

cashunits/additionalProperties/cashInCount

Count of items that have entered the cash unit. This counter is incremented whenever an item enters a cash unit for any reason, unless it originated from this cash unit but was

returned without being accessible to a customer. For a retract cash unit this value represents the total number of items of all types in the cash unit, or if the device cannot count items during a retract operation this value will be zero. This value is persistent.

cashunits/additionalProperties/noteNumberList

Array of cash items inside the cash unit. The content of this structure is persistent. If the cash unit is Dispenser specific cash unit with *type billCassette* or the contents of the cash unit are not known this structure will be omitted. If the cash unit is of *type retractCassette* this pointer will be omitted except for the following cases:

- If the retract cash unit is configured to accept level 2 notes then the number and type of level 2 notes is returned in the *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of level 2 notes.
- If items are recognized during retract operations then the number and type of notes retracted is returned in *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of retracted items.

cashunits/additionalProperties/noteNumberList/noteNumber

Array of banknote numbers the cash unit contains.

cashunits/additionalProperties/noteNumberList/noteNumber/noteID

Identification of note type. The Note ID represents the note identifiers reported by the *CashAcceptor.BanknoteTypes* command. If this value is zero then the note type is unknown.

cashunits/additionalProperties/noteNumberList/noteNumber/count

Actual count of cash items. The value is incremented each time cash items are moved to a cash unit. In the case of recycle cash units this count is decremented as defined in the description of the *logicalCount* field.

cashunits/additionalProperties/noteIDs

Array of integers which contains the note IDs of the banknotes the cash-in cash unit or recycle cash unit can take. This field only applies to *individual* cassette types. If there are no note IDs defined for the cassette or the cassette is not defined as *individual* then *noteIDs* will be omitted.

Event Messages

None

4.3.2 - CashManagement.GetTellerInfo

This command only applies to Teller devices. It allows the client to obtain counts for each currency assigned to the teller. These counts represent the total amount of currency dispensed by the teller in all transactions.

This command also enables the client to obtain the position assigned to each teller. If the input parameter is NULL, this command will return information for all tellers and all currencies. The teller information is persistent.

Command Message

| Payload | Type | Required |
|-----------------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "tellerID": 0, | integer | |
| "currencyID": Add example to YAML | string | |
| } | | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0**tellerID**

Identification of the teller. If the value of *tellerID* is not valid the error *invalidTellerID* is reported.

currencyID

Three character ISO format currency identifier [Ref 2].

Completion Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "invalidCurrency", | string | |
| "tellerDetails": [{ | array (object) | |
| "tellerID": 0, | integer | |
| "inputPosition": "none", | string | |
| "outputPosition": "none", | string | |
| "tellerTotals": { | object | |
| "additionalProperties": { | object | |
| "itemsReceived": 0, | number | |
| "itemsDispensed": 0, | number | |
| "coinsReceived": 0, | number | |
| "coinsDispensed": 0, | number | |
| "cashBoxReceived": 0, | number | |
| "cashBoxDispensed": 0 | number | |
| } | | |
| } | | |
| }] | | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

- invalidCurrency - Specified currency not currently available.
- invalidTellerId - Invalid teller ID.

tellerDetails

Array of teller detail objects.

tellerDetails/tellerID

Identification of the teller.

tellerDetails/inputPosition

The input position assigned to the teller for cash entry. Following values are possible:

- none - No position is assigned to the teller.
- left - Left position is assigned to the teller.
- right - Right position is assigned to the teller.
- center - Center position is assigned to the teller.
- top - Top position is assigned to the teller.
- bottom - Bottom position is assigned to the teller.
- front - Front position is assigned to the teller.
- rear - Rear position is assigned to the teller.

tellerDetails/outputPosition

The output position from which cash is presented to the teller. Following values are possible:

- none - No position is assigned to the teller.

- left - Left position is assigned to the teller.
- right - Right position is assigned to the teller.
- center - Center position is assigned to the teller.
- top - Top position is assigned to the teller.
- bottom - Bottom position is assigned to the teller.
- front - Front position is assigned to the teller.
- rear - Rear position is assigned to the teller.

tellerDetails/tellerTotals

List of teller total objects. There is one object per currency.

tellerDetails/tellerTotals/additionalProperties

The object name is the three character ISO format currency identifier [Ref 2].

tellerDetails/tellerTotals/additionalProperties/itemsReceived

The total amount of items (other than coins) of the specified currency accepted. The amount is expressed as floating point value.

tellerDetails/tellerTotals/additionalProperties/itemsDispensed

The total amount of items (other than coins) of the specified currency dispensed. The amount is expressed as floating point value.

tellerDetails/tellerTotals/additionalProperties/coinsReceived

The total amount of coin currency accepted. The amount is expressed as floating point value.

tellerDetails/tellerTotals/additionalProperties/coinsDispensed

The total amount of coin currency dispensed. The amount is expressed as floating point value.

tellerDetails/tellerTotals/additionalProperties/cashBoxReceived

The total amount of cash box currency accepted. The amount is expressed as floating point value.

tellerDetails/tellerTotals/additionalProperties/cashBoxDispensed

The total amount of cash box currency dispensed. The amount is expressed as floating point value.

Event Messages

None

4.3.3 - CashManagement.GetItemInfo

This command is used to get information about detected items. It can be used to get information about individual items, all items of a certain level, or all items that have information available. This information is available from the point where the first CashManagement.InfoAvailableEvent event is generated until one of the following commands is executed:

CashAcceptor.CashInStart, CashAcceptor.CashIn, CashAcceptor.CashInRollback,
 CashAcceptor.CashInEnd, CashAcceptor.Retract, CashAcceptor.Reset,
 CashAcceptor.CreateP6Signature, CashAcceptor.Replenish, CashAcceptor.CashUnitCount.
 Dispenser.Dispense, Dispenser.Count, Dispenser.Present, Dispenser.Retract,
 Dispenser.Reject, Dispenser.OpenShutter, Dispenser.CloseShutter, Dispenser.Reset,
 CashManagement.StartExchange, CashManagement.EndExchange,
 CashManagement.CalibrateCashUnit, Dispenser.TestCashUnits.

Command Message

| Payload | Type | Required |
|------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "level": "level1", | string | |
| "index": 0, | integer | |
| "itemInfoType": { | object | |
| "serialNumber": false, | boolean | |
| "signature": false, | boolean | |
| "imageFile": false | boolean | |

}

}

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

level

Defines the requested note level. Following values are possible:

- level1 - Information for level 1 notes. Only an image file can be retrieved for level 1 notes.
- level2 - Information for level 2 notes. On systems that do not classify notes as level 2 this value cannot be used and invalidData will be returned.
- level3 - Information for level 3 notes. On systems that do not classify notes as level 3 this value cannot be used and invalidData will be returned.
- level4 - Information for level 4 notes.
- levelAll - Information for all levels and all items is to be returned with the *itemsList* output parameter.

index

Specifies the index for the item information required. If no index is provided, all items of the specified **level** will be returned. If *level* is set to levelAll, this property will be ignored.

itemInfoType

Specifies the type of information required. If nothing is specified, all available information will be returned.

itemInfoType/serialNumber

Serial number of the item.

itemInfoType/signature

Signature of the item.

itemInfoType/imageFile

Image file of the item.

Completion Message

| Payload | Type | Required |
|---|----------------|-----------------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "itemsList": [{ | array (object) | |
| "level": "level1", | string | |
| "serialNumber": Add example to YAML, | string | |
| "orientation": { | object | |
| "frontTop": false, | boolean | |
| "frontBottom": false, | boolean | |
| "backTop": false, | boolean | |
| "backBottom": false, | boolean | |
| "unknown": false, | boolean | |
| "notSupported": false | boolean | |
| }, | | |
| "p6Signature": Add example to YAML, | string | |
| "imageFile": Add example to YAML, | string | |
| "onClassificationList": "onClassificationList", | string | |
| "itemLocation": "device", | string | |
| "cashunit": Add example to YAML, | string | |

```
"itemDeviceLocation": "stacker"          string  
}  
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

itemsList

Array of "item info" objects.

itemsList/level

Defines the note level. Following values are possible:

- level1 - A level 1 banknote.
- level2 - A level 2 banknote.
- level3 - A level 3 banknote.
- level4Fit - A fit level 4 banknote.
- level4Unfit - An unfit level 4 banknote.

itemsList/serialNumber

This field contains the serial number of the item as a string. A '?' character (0x003F) is used to represent any serial number character that cannot be recognized. If no serial number is available or has not been requested then *serialNumber* is omitted.

itemsList/orientation

Orientation of the entered banknote.

itemsList/orientation/frontTop

If note is inserted wide side as the leading edge, the note was inserted with the front image facing up and the top edge of the note was inserted first. If the note is inserted short side as the leading edge, the note was inserted with the front image face up and the left edge was inserted first.

itemsList/orientation/frontBottom

If note is inserted wide side as the leading edge, the note was inserted with the front image facing up and the bottom edge of the note was inserted first. If the note is inserted short side as the leading edge, the note was inserted with the front image face up and the right edge was inserted first.

itemsList/orientation/backTop

If note is inserted wide side as the leading edge, the note was inserted with the back image facing up and the top edge of the note was inserted first. If the note is inserted short side as the leading edge, the note was inserted with the back image face up and the left edge was inserted first.

itemsList/orientation/backBottom

If note is inserted wide side as the leading edge, the note was inserted with the back image facing up and the bottom edge of the note was inserted first. If the note is inserted short side as the leading edge, the note was inserted with the back image face up and the right edge was inserted first.

itemsList/orientation/unknown

The orientation for the inserted note can not be determined.

itemsList/orientation/notSupported

The hardware is not capable to determine the orientation.

itemsList/p6Signature

Base64 encoded binary file containing only the vendor specific P6 signature data. If no P6 signature is available then this field is omitted.

itemsList/imageFile

Base64 encoded binary image data. If the Service does not support this function or the image file has not been requested then *imageFile* is omitted.

itemsList/onClassificationList

Specifies if the serial number reported in the *serialNumber* field is on the classification list. If the classification list reporting capability is not supported this field will be omitted. Following values are possible:

- *onClassificationList* - The serial number of the items is on the classification list.
- *notOnClassificationList* - The serial number of the items is not on the classification list.
- *classificationListUnknown* - It is unknown if the serial number of the item is on the classification list.

itemsList/itemLocation

Specifies the location of the item. Following values are possible:

- *device* - The item is inside the device in some position other than a cash unit.
- *cashUnit* - The item is in a cash unit. The cash unit is defined by [cashunit](#).
- *customer* - The item has been dispensed to the customer.
- *unknown* - The item location is unknown.

itemsList/cashunit

If [itemLocation](#) is *cashUnit* this parameter specifies the object name of the cash unit which received the item as stated by the [CashManagement.GetCashUnitInfo](#) command. If *itemLocation* is not *cashUnit* *cashunit* will be omitted.

itemsList/itemDeviceLocation

If [itemLocation](#) is *device* this parameter specifies where the item is in the device. If *itemLocation* is not *device* then *itemDeviceLocation* will be omitted. Following values are possible:

- *stacker* - The item is in the intermediate stacker.
- *output* - The item is at the output position. The items have not been in customer access.
- *transport* - The item is at another location in the device.
- *unknown* - The item is in the device but its location is unknown.

Event Messages

None

4.3.4 - CashManagement.GetClassificationList

This command is used to retrieve the entire note classification information pre-set inside the device or set via the CashManagement.SetClassificationList command. This provides the functionality to blacklist notes and allows additional flexibility, for example to specify that notes can be taken out of circulation by specifying them as unfit. Any items not returned in this list will be handled according to normal classification rules.

Command Message

| Payload | Type | Required |
|--------------------------------------|------|----------|
| { "timeout": 5000 integer } | | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "version": Add example to YAML, | string | |

```
"classificationElements": [{  
    "array (object)  
    "serialNumber": Add example to YAML,  
    string  
    "currencyID": Add example to YAML,  
    string  
    "value": 0,  
    number  
    "level": "level1"  
    string  
}  
]  
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

version

This is a client defined string that sets the version identifier of the classification list. This property can be omitted if it has no version identifier.

classificationElements

Array of classification objects.

classificationElements/serialNumber

This string defines the serial number or a mask of serial numbers of one element with the defined currency and value. For a definition of the mask see Section Note Classification.

classificationElements/currencyID

The three character ISO 4217 format currency identifier [Ref. 2] of the element.

classificationElements/value

The value of the element. This field can be zero to represent all values.

classificationElements/level

Specifies the note level. Following values are possible:

- level1 - The element specifies notes to be treated as level 1 notes.
- level2 - The element specifies notes to be treated as level 2 notes.
- level3 - The element specifies notes to be treated as level 3 notes.
- level4Fit - The element specifies notes to be treated as fit level 4 notes.
- level4Unfit - The element specifies notes to be treated as unfit level 4 notes.

Event Messages

None

4.3.5 - CashManagement.SetTellerInfo

This command allows the client to initialize counts for each currency assigned to the teller. The values set by this command are persistent. This command only applies to Teller ATMs.

Command Message

| Payload | Type | Required |
|---------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "action": "createTeller", | string | |
| "tellerDetails": { | object | |
| "tellerID": 0, | integer | |
| "inputPosition": "none", | string | |
| "outputPosition": "none", | string | |
| "tellerTotals": { | object | |

```
"additionalProperties": { object  
  "itemsReceived": 0,      number  
  "itemsDispensed": 0,    number  
  "coinsReceived": 0,     number  
  "coinsDispensed": 0,    number  
  "cashBoxReceived": 0,   number  
  "cashBoxDispensed": 0,  number  
}  
}  
}  
}
```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

action

The action to be performed. Following values are possible:

- `createTeller` - A teller is to be added.
- `modifyTeller` - Information about an existing teller is to be modified.
- `deleteTeller` - A teller is to be removed.

tellerDetails

Teller details object.

tellerDetails/tellerID

Identification of the teller.

tellerDetails/inputPosition

The input position assigned to the teller for cash entry. Following values are possible:

- none - No position is assigned to the teller.
- left - Left position is assigned to the teller.
- right - Right position is assigned to the teller.
- center - Center position is assigned to the teller.
- top - Top position is assigned to the teller.
- bottom - Bottom position is assigned to the teller.
- front - Front position is assigned to the teller.
- rear - Rear position is assigned to the teller.

tellerDetails/outputPosition

The output position from which cash is presented to the teller. Following values are possible:

- none - No position is assigned to the teller.
- left - Left position is assigned to the teller.
- right - Right position is assigned to the teller.
- center - Center position is assigned to the teller.
- top - Top position is assigned to the teller.
- bottom - Bottom position is assigned to the teller.
- front - Front position is assigned to the teller.
- rear - Rear position is assigned to the teller.

tellerDetails/tellerTotals

List of teller total objects. There is one object per currency.

tellerDetails/tellerTotals/additionalProperties

The object name is the three character ISO format currency identifier [Ref 2].

tellerDetails/tellerTotals/additionalProperties/itemsReceived

The total amount of items (other than coins) of the specified currency accepted. The amount is expressed as floating point value.

tellerDetails/tellerTotals/additionalProperties/itemsDispensed

The total amount of items (other than coins) of the specified currency dispensed. The amount is expressed as floating point value.

tellerDetails/tellerTotals/additionalProperties/coinsReceived

The total amount of coin currency accepted. The amount is expressed as floating point value.

tellerDetails/tellerTotals/additionalProperties/coinsDispensed

The total amount of coin currency dispensed. The amount is expressed as floating point value.

tellerDetails/tellerTotals/additionalProperties/cashBoxReceived

The total amount of cash box currency accepted. The amount is expressed as floating point value.

tellerDetails/tellerTotals/additionalProperties/cashBoxDispensed

The total amount of cash box currency dispensed. The amount is expressed as floating point value.

Completion Message

| <u>Payload</u> | <u>Type</u> | <u>Required</u> |
|--|-------------|-----------------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "invalidCurrency" | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

- invalidCurrency - The specified currency is not currently available.
- invalidTellerId - The teller ID is invalid.
- unsupportedPosition - The position specified is not supported.
- exchangeActive - The target teller is currently in the middle of an exchange operation.

Event Messages

- [CashManagement.TellerInfoChangedEvent](#)

4.3.6 - [CashManagement.SetCashUnitInfo](#)

This command is used to adjust information about the status and contents of the cash units present in the device. Only fields that are to be changed need to be set in the payload of this command. Values that are not meant to change, can be ommitted.

This command generates the [CashManagement.CashUnitInfoChangedEvent](#) to inform clients that cash unit information has been changed.

This command can be used to change software counters, thresholds and the client lock. Other fields in the input structure will only take effect, if used in exchange state.

The following fields of the structure may be updated by this command outside of the exchange state:

count

logicalCount

cashInCount

maximum

appLock

noteNumberList (*contents must be consistent with ulCount*)

initialCount
dispensedCount
presentedCount
retractedCount
rejectCount
minimum

Any other changes must be performed while in an exchange state.

The values set by this command are persistent.

Command Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "cashunits": { | object | |
| "additionalProperties": { | object | |
| "type": "billCassette", | string | |
| "currencyID": Add example to YAML, | string | |
| "value": 0, | number | |
| "logicalCount": 0, | integer | |
| "maximum": 0, | integer | |
| "appLock": false, | boolean | |
| "cashUnitName": Add example to YAML, | string | |
| "initialCount": 0, | integer | |
| "dispensedCount": 0, | integer | |
| "presentedCount": 0, | integer | |
| "retractedCount": 0, | integer | |
| "rejectCount": 0, | integer | |
| "minimum": 0, | integer | |
| "physicalPositionName": Add example to YAML, | string | |

```

"unitID": Add example to YAML,           string
"count": 0,                            integer
"maximumCapacity": 0,                  integer
"hardwareSensor": false,              boolean
"itemType": {                           object
  "all": false,                         boolean
  "unfit": false,                       boolean
  "individual": false,                  boolean
  "level1": false,                      boolean
  "level2": false,                      boolean
  "level3": false,                      boolean
  "itemProcessor": false,               boolean
  "unfitIndividual": false            boolean
},
"cashInCount": 0,                      integer
"noteNumberList": {                    object
  "noteNumber": [{                     array (object)
    "noteID": 0,                        integer
    "count": 0                          integer
  }]
},
"noteIDs": [0]                         array (integer)
}
}
}

```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

cashunits

Object containing cash unit settings.

cashunits/additionalProperties

Cash unit settings.

cashunits/additionalProperties/type

Type of cash unit. Following values are possible:

- notApplicable - Not applicable. Typically means cash unit is missing.
- rejectCassette - Reject cash unit. This type will also indicate a combined reject/retract cash unit.
- billCassette - Cash unit containing bills.
- coinCylinder - Coin cylinder.
- coinDispenser - Coin dispenser as a whole unit.
- retractCassette - Retract cash unit.
- coupon - Cash unit containing coupons or advertising material.
- document - Cash unit containing documents.
- replenishmentContainer - Replenishment container. A cash unit can be refilled from a replenishment container.
- recycling - Recycling cash unit. This unit is only present when the device implements the Dispenser and CashAcceptor interfaces.
- cashIn - Cash-in cash unit.

cashunits/additionalProperties/currencyID

A three character string storing the ISO format [Ref. 2] Currency ID. This value will be omitted for cash units which contain items of more than one currency type or items to which currency is not applicable. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

cashunits/additionalProperties/value

Supplies the value of a single item in the cash unit. This value is expressed as floating point value. If the *currencyID* field for this cash unit is omitted, then this field will contain zero. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

cashunits/additionalProperties/logicalCount

The meaning of this count depends on the type of cash unit. This value is persistent. For all cash units except retract cash units (*type* is not *retractCassette*) this value specifies the number of items inside the cash unit. For all dispensing cash units (*type* is *billCassette*, *coinCylinder*, *coinDispenser*, *coupon*, *document* or *recycling*), this value includes any items from the cash unit not yet presented to the customer. This count is only decremented when the items are either known to be in customer access or successfully rejected. If the cash unit is usable from the CashAcceptor interface (*type* is *recycling*, *cashIn*, *retractCassette* or *rejectCassette*) then this value will be incremented as a result of a cash-in operation. Note that for a reject cash unit (*type* is *rejectCassette*), this value is unreliable, since the typical reason for dumping items to the reject cash unit is a suspected count failure. For a retract cash unit (*type* is *retractCassette*) this value specifies the number of retract operations which result in items entering the cash unit.

cashunits/additionalProperties/maximum

When *count* reaches this value the threshold event

`CashManagement.CashUnitThresholdEvent (high)` will be generated. This value can be different from the actual capacity of the cassette. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

cashunits/additionalProperties/appLock

If this value is TRUE items cannot be dispensed from or deposited into the cash unit. If this value is TRUE and the client attempts to use the cash unit a `CashManagement.CashUnitErrorEvent` event will be generated and an error completion message will be returned. This value is persistent.

cashunits/additionalProperties/cashUnitName

A name which helps to identify the type of the cash unit. This is especially useful in the case of cash units of type *document* where different documents can have the same currency and value. For example, travelers checks and bank checks may have the same currency and value but still need to be identifiable as different types of document. Where this value is not

relevant (e.g. in bill cash units) the property can be omitted. This value is persistent.

cashunits/additionalProperties/initialCount

Initial number of items contained in the cash unit. This value is persistent.

cashunits/additionalProperties/dispensedCount

The number of items dispensed from this cash unit. This count is incremented when the items are removed from the cash units. This count includes any items that were rejected during the dispense operation and are no longer in this cash unit. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

cashunits/additionalProperties/presentedCount

The number of items from this cash unit that have been presented to the customer. This count is incremented when the items are presented to the customer. If it is unknown if a customer has been presented with the items, then this count is not updated. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

cashunits/additionalProperties/retractedCount

The number of items that have been accessible to a customer and retracted into the cash unit. This value is persistent.

cashunits/additionalProperties/rejectCount

The number of items dispensed from this cash unit which have been rejected, are in a cash unit other than this cash unit, and which have not been accessible to a customer. This value may be unreliable, since a typical reason for rejecting items is a suspected pick failure. Other reasons for rejecting items may include incorrect note denominations, classifications not valid for dispensing, or where the transaction has been cancelled and a Reject command has been called. For reject and retract cash units (*type* is *rejectCassette* or *retractCassette*) this field does not apply and will be reported as zero. This value is persistent.

cashunits/additionalProperties/minimum

This field is not applicable to retract and reject cash units. For all cash units which dispense items (all other), when *count* reaches this value the threshold event

CashManagement.CashUnitThresholdEvent (*low*) will be generated. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

cashunits/additionalProperties/physicalPositionName

A name identifying the physical location of the cash unit.

cashunits/additionalProperties/unitID

A 5 character string uniquely identifying the cash unit.

cashunits/additionalProperties/count

As defined by the *logicalCount* description, but with the following exceptions: This count does not include items dispensed but not yet presented. On cash units with *type* set to "retractCassette" the count represents the number of items, unless the device cannot count items during a retract, in which case this count will be zero. This value is persistent.

cashunits/additionalProperties/maximumCapacity

The maximum number of items the cash unit can hold. This is only for informational purposes. No threshold event CashManagement.CashUnitThresholdEvent will be generated. This value is persistent.

cashunits/additionalProperties/hardwareSensor

Specifies whether or not threshold events can be generated based on hardware sensors in the device. If this value is TRUE then threshold events may be generated based on hardware sensors as opposed to counts.

cashunits/additionalProperties/itemType

Specifies the type of items the cash unit takes as a combination of the following flags. The table in the Comments section of this command defines how to interpret the combination of these flags (TODO: include Table)

cashunits/additionalProperties/itemType/all

The cash unit takes all fit banknote types. These are level 4 notes which are fit for recycling.

`cashunits/additionalProperties/itemType/unfit`

The cash unit takes all unfit banknotes. These are level 4 notes which are unfit for recycling.

`cashunits/additionalProperties/itemType/individual`

The cash unit takes all types of fit banknotes specified in an individual list. These are level 4 notes which are fit for recycling.

`cashunits/additionalProperties/itemType/level1`

Level 1 note types are stored in this cash unit.

`cashunits/additionalProperties/itemType/level2`

If notes can be classified as level 2, then level 2 note types are stored in this cash unit.

`cashunits/additionalProperties/itemType/level3`

If notes can be classified as level 3, then level 3 note types are stored in this cash unit.

`cashunits/additionalProperties/itemType/itemProcessor`

The cash unit can accept items on the ItemProcessor interface.

`cashunits/additionalProperties/itemType/unfitIndividual`

The cash unit takes all types of unfit banknotes specified in an individual list. These are level 4 notes which are unfit for recycling.

`cashunits/additionalProperties/cashInCount`

Count of items that have entered the cash unit. This counter is incremented whenever an item enters a cash unit for any reason, unless it originated from this cash unit but was returned without being accessible to a customer. For a retract cash unit this value represents the total number of items of all types in the cash unit, or if the device cannot

count items during a retract operation this value will be zero. This value is persistent.

cashunits/additionalProperties/noteNumberList

Array of cash items inside the cash unit. The content of this structure is persistent. If the cash unit is Dispenser specific cash unit with *type billCassette* or the contents of the cash unit are not known this structure will be omitted. If the cash unit is of *type retractCassette* this pointer will be omitted except for the following cases:

- If the retract cash unit is configured to accept level 2 notes then the number and type of level 2 notes is returned in the *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of level 2 notes.
- If items are recognized during retract operations then the number and type of notes retracted is returned in *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of retracted items.

cashunits/additionalProperties/noteNumberList/noteNumber

Array of banknote numbers the cash unit contains.

cashunits/additionalProperties/noteNumberList/noteNumber/noteID

Identification of note type. The Note ID represents the note identifiers reported by the *CashAcceptor.BanknoteTypes* command. If this value is zero then the note type is unknown.

cashunits/additionalProperties/noteNumberList/noteNumber/count

Actual count of cash items. The value is incremented each time cash items are moved to a cash unit. In the case of recycle cash units this count is decremented as defined in the description of the *logicalCount* field.

cashunits/additionalProperties/noteIDs

Array of integers which contains the note IDs of the banknotes the cash-in cash unit or recycle cash unit can take. This field only applies to *individual* cassette types. If there are no note IDs defined for the cassette or the cassette is not defined as *individual* then *noteIDs* will be omitted.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "invalidTellerId" | string | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

- invalidTellerId - Invalid teller ID. This error will never be generated by a Self-Service device.
- invalidCashUnit - Invalid cash unit.
- noExchangeActive - The device is not in an exchange state. The command can only be completed in exchange state.
- cashUnitError - A problem occurred with a cash unit. A CashManagement.CashUnitErrorEvent will be posted with the details.

Event Messages

- [CashManagement.CashUnitThresholdEvent](#)
- [CashManagement.CashUnitInfoChangedEvent](#)
- [CashManagement.CashUnitErrorEvent](#)

4.3.7 - CashManagement.OpenSafeDoor

This command unlocks the safe door or starts the time delay count down prior to unlocking the safe door, if the device supports it. The command completes when the door is unlocked or the timer has started.

Command Message

| Payload | Type | Required |
|---------------------------|---------|----------|
| { "timeout": 5000 } | integer | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "exchangeActive" | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

- exchangeActive - The device is in an exchange state.

Event Messages

None

4.3.8 - CashManagement.StartExchange

This command puts the device in an exchange state, i.e. a state in which cash units can be emptied, replenished, removed or replaced. Other than the updates which can be made via the CashManagement.SetCashUnitInfo command, outside of an exchange state, all changes to a cash unit must take place while the cash unit is in an exchange state.

The command returns current cash unit information in the form described in the documentation of the CashManagement.CashUnitInfo command. This command will also initiate any physical processes which may be necessary to make the cash units accessible. Before using this command a client should first have obtained exclusive control of the CashManagement interface.

This command may return a successful completion even if CashManagement.CashUnitErrorEvents are generated. If this command returns a successful completion the device is in an exchange state.

In Exchange state the CashManagement.SetCashUnitInfo command can be used multiple times to adjust the cash unit information, until the CashManagement.EndExchange command is performed.

While in an exchange state the device will process all requests, excluding cash related commands other than CashManagement.EndExchange, Dispenser.SetMixTable and Reset commands (e.g. Dispenser.Reset).

Any other command will result in the error "exchangeActive" being generated.

If an error is returned by this command, the response will include current cash unit information.

Command Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "exchangeType": "byHand", | string | |
| "tellerID": 0, | integer | |
| "cashunitList": [Add example to YAML], | array (string) | |
| "output": { | object | |
| "cashunit": Add example to YAML, | string | |
| "position": { | object | |
| "default": false, | boolean | |
| "left": false, | boolean | |
| "right": false, | boolean | |
| "center": false, | boolean | |
| "top": false, | boolean | |
| "bottom": false, | boolean | |
| "front": false, | boolean | |
| "rear": false | boolean | |
| }, | | |
| "targetCashunit": Add example to YAML | string | |
| } | | |
| } | | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not

timeout but can be cancelled.

default: 0

exchangeType

Specifies the type of cash unit exchange operation. Following values are possible:

- byHand - The cash units will be replenished manually either by filling or emptying the cash unit by hand or by replacing the cash unit.
- toCassettes - Items will be moved from the replenishment container to the bill cash units.
- clearRecycler - Items will be moved from a recycle cash unit to a cash unit or output position.
- depositInto - Items will be moved from the deposit entrance to the bill cash units. See section x.y (TODO) for an example flow.

tellerID

Identifies the teller. If the device is a Self-Service ATM this field is ignored.

cashunitList

Array of strings containing the object names of the cash units to be exchanged as stated by the [CashManagement.GetCashUnitInfo](#) command.

output

This field is used when the [exchangeType](#) is clearRecycler, i.e. a recycle cash unit is to be emptied.

output/cashunit

Object name of recycle cash unit to be emptied as stated by the [CashManagement.GetCashUnitInfo](#) command.

output/position

Determines to which position the cash should be moved as a combination of the following flags:

output/position/default

Move items to a cash unit. If no cash unit is specified in *targetNumber*, use the default output position.

output/position/left

Move items to the left output position.

output/position/right

Move items to the right output position.

output/position/center

Move items to the center output position.

output/position/top

Move items to the top output position.

output/position/bottom

Move items to the bottom output position.

output/position/front

Move items to the front output position.

output/position/rear

Move items to the rear output position.

output/targetCashunit

Object name of the cash unit the items are to be moved to as stated by the [CashManagement.GetCashUnitInfo](#) command.

Completion Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "invalidCurrency", | string | |
| "cashunits": { | object | |
| "additionalProperties": { | object | |
| "status": "ok", | string | |
| "type": "billCassette", | string | |
| "currencyID": Add example to YAML, | string | |
| "value": 0, | number | |
| "logicalCount": 0, | integer | |
| "maximum": 0, | integer | |
| "appLock": false, | boolean | |
| "cashUnitName": Add example to YAML, | string | |
| "initialCount": 0, | integer | |
| "dispensedCount": 0, | integer | |
| "presentedCount": 0, | integer | |
| "retractedCount": 0, | integer | |
| "rejectCount": 0, | integer | |
| "minimum": 0, | integer | |
| "physicalPositionName": Add example to YAML, | string | |
| "unitID": Add example to YAML, | string | |
| "count": 0, | integer | |
| "maximumCapacity": 0, | integer | |
| "hardwareSensor": false, | boolean | |

```

"itemType": {                                     object
  "all": false,                                boolean
  "unfit": false,                               boolean
  "individual": false,                          boolean
  "level1": false,                             boolean
  "level2": false,                             boolean
  "level3": false,                             boolean
  "itemProcessor": false,                      boolean
  "unfitIndividual": false                    boolean
},
"cashInCount": 0,                                integer
"noteNumberList": {                               object
  "noteNumber": [{                                array (object)
    "noteID": 0,                                 integer
    "count": 0,                                 integer
  }]
},
"noteIDs": [0]                                    array (integer)
}
}
}

```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

- invalidTellerId - Invalid teller ID. This error will never be generated by a Self-Service device.
- cashUnitError - An error occurred with a cash unit while performing the exchange operation. A CashManagement.CashUnitErrorEvent will be sent with the details.
- tooManyItems - This error is generated if the contents of the recycle cash unit cannot be completely emptied to the output position. The maximum possible number of items is moved to the output position.
- exchangeActive - The device is already in an exchange state.
- cashInActive - A cash-in transaction is active.

cashunits

Object containing cash unit information.

cashunits/additionalProperties

Cash unit information.

cashunits/additionalProperties/status

Supplies the status of the cash unit. Following values are possible:

- ok - The cash unit is in a good state.
- full - The cash unit is full.
- high - The cash unit is almost full (i.e. reached or exceeded the threshold defined by *maximum*).
- low - The cash unit is almost empty (i.e. reached or below the threshold defined by *minimum*).
- empty - The cash unit is empty, or insufficient items in the cash unit are preventing further dispense operations.
- inoperative - The cash unit is inoperative.
- missing - The cash unit is missing.
- noValue - The values of the specified cash unit are not available.
- noReference - There is no reference value available for the notes in this cash unit.

The cash unit has not been calibrated.

- manipulated - The cash unit has been inserted (including removal followed by a reinsertion) when the device was not in the exchange state. This cash unit cannot be dispensed from.

cashunits/additionalProperties/type

Type of cash unit. Following values are possible:

- notApplicable - Not applicable. Typically means cash unit is missing.
- rejectCassette - Reject cash unit. This type will also indicate a combined reject/retract cash unit.
- billCassette - Cash unit containing bills.
- coinCylinder - Coin cylinder.
- coinDispenser - Coin dispenser as a whole unit.
- retractCassette - Retract cash unit.
- coupon - Cash unit containing coupons or advertising material.
- document - Cash unit containing documents.
- replenishmentContainer - Replenishment container. A cash unit can be refilled from a replenishment container.
- recycling - Recycling cash unit. This unit is only present when the device implements the Dispenser and CashAcceptor interfaces.
- cashIn - Cash-in cash unit.

cashunits/additionalProperties/currencyID

A three character string storing the ISO format [Ref. 2] Currency ID. This value will be omitted for cash units which contain items of more than one currency type or items to which currency is not applicable. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

cashunits/additionalProperties/value

Supplies the value of a single item in the cash unit. This value is expressed as floating point value. If the *currencyID* field for this cash unit is omitted, then this field will contain zero. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

cashunits/additionalProperties/logicalCount

The meaning of this count depends on the type of cash unit. This value is persistent. For all cash units except retract cash units (*type* is not *retractCassette*) this value specifies the number of items inside the cash unit. For all dispensing cash units (*type* is *billCassette*,

coinCylinder, coinDispenser, coupon, document or recycling), this value includes any items from the cash unit not yet presented to the customer. This count is only decremented when the items are either known to be in customer access or successfully rejected. If the cash unit is usable from the CashAcceptor interface (*type* is *recycling*, *cashIn*, *retractCassette* or *rejectCassette*) then this value will be incremented as a result of a cash-in operation. Note that for a reject cash unit (*type* is *rejectCassette*), this value is unreliable, since the typical reason for dumping items to the reject cash unit is a suspected count failure. For a retract cash unit (*type* is *retractCassette*) this value specifies the number of retract operations which result in items entering the cash unit.

cashunits/additionalProperties/maximum

When *count* reaches this value the threshold event

CashManagement.CashUnitThresholdEvent (*high*) will be generated. This value can be different from the actual capacity of the cassette. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

cashunits/additionalProperties/appLock

If this value is TRUE items cannot be dispensed from or deposited into the cash unit. If this value is TRUE and the client attempts to use the cash unit a CashManagement.CashUnitErrorEvent event will be generated and an error completion message will be returned. This value is persistent.

cashunits/additionalProperties/cashUnitName

A name which helps to identify the type of the cash unit. This is especially useful in the case of cash units of type *document* where different documents can have the same currency and value. For example, travelers checks and bank checks may have the same currency and value but still need to be identifiable as different types of document. Where this value is not relevant (e.g. in bill cash units) the property can be omitted. This value is persistent.

cashunits/additionalProperties/initialCount

Initial number of items contained in the cash unit. This value is persistent.

cashunits/additionalProperties/dispensedCount

The number of items dispensed from this cash unit. This count is incremented when the items are removed from the cash units. This count includes any items that were rejected during the dispense operation and are no longer in this cash unit. This field is always zero

for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

cashunits/additionalProperties/presentedCount

The number of items from this cash unit that have been presented to the customer. This count is incremented when the items are presented to the customer. If it is unknown if a customer has been presented with the items, then this count is not updated. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

cashunits/additionalProperties/retractedCount

The number of items that have been accessible to a customer and retracted into the cash unit. This value is persistent.

cashunits/additionalProperties/rejectCount

The number of items dispensed from this cash unit which have been rejected, are in a cash unit other than this cash unit, and which have not been accessible to a customer. This value may be unreliable, since a typical reason for rejecting items is a suspected pick failure. Other reasons for rejecting items may include incorrect note denominations, classifications not valid for dispensing, or where the transaction has been cancelled and a Reject command has been called. For reject and retract cash units (*type* is *rejectCassette* or *retractCassette*) this field does not apply and will be reported as zero. This value is persistent.

cashunits/additionalProperties/minimum

This field is not applicable to retract and reject cash units. For all cash units which dispense items (all other), when *count* reaches this value the threshold event CashManagement.CashUnitThresholdEvent (*low*) will be generated. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

cashunits/additionalProperties/physicalPositionName

A name identifying the physical location of the cash unit.

cashunits/additionalProperties/unitID

A 5 character string uniquely identifying the cash unit.

cashunits/additionalProperties/count

As defined by the *logicalCount* description, but with the following exceptions: This count does not include items dispensed but not yet presented. On cash units with *type* set to "retractCassette" the count represents the number of items, unless the device cannot count items during a retract, in which case this count will be zero. This value is persistent.

cashunits/additionalProperties/maximumCapacity

The maximum number of items the cash unit can hold. This is only for informational purposes. No threshold event CashManagement.CashUnitThresholdEvent will be generated. This value is persistent.

cashunits/additionalProperties/hardwareSensor

Specifies whether or not threshold events can be generated based on hardware sensors in the device. If this value is TRUE then threshold events may be generated based on hardware sensors as opposed to counts.

cashunits/additionalProperties/itemType

Specifies the type of items the cash unit takes as a combination of the following flags. The table in the Comments section of this command defines how to interpret the combination of these flags (TODO: include Table)

cashunits/additionalProperties/itemType/all

The cash unit takes all fit banknote types. These are level 4 notes which are fit for recycling.

cashunits/additionalProperties/itemType/unfit

The cash unit takes all unfit banknotes. These are level 4 notes which are unfit for recycling.

cashunits/additionalProperties/itemType/individual

The cash unit takes all types of fit banknotes specified in an individual list. These are level 4 notes which are fit for recycling.

cashunits/additionalProperties/itemType/level1

Level 1 note types are stored in this cash unit.

cashunits/additionalProperties/itemType/level2

If notes can be classified as level 2, then level 2 note types are stored in this cash unit.

cashunits/additionalProperties/itemType/level3

If notes can be classified as level 3, then level 3 note types are stored in this cash unit.

cashunits/additionalProperties/itemType/itemProcessor

The cash unit can accept items on the ItemProcessor interface.

cashunits/additionalProperties/itemType/unfitIndividual

The cash unit takes all types of unfit banknotes specified in an individual list. These are level 4 notes which are unfit for recycling.

cashunits/additionalProperties/cashInCount

Count of items that have entered the cash unit. This counter is incremented whenever an item enters a cash unit for any reason, unless it originated from this cash unit but was returned without being accessible to a customer. For a retract cash unit this value represents the total number of items of all types in the cash unit, or if the device cannot count items during a retract operation this value will be zero. This value is persistent.

cashunits/additionalProperties/noteNumberList

Array of cash items inside the cash unit. The content of this structure is persistent. If the cash unit is Dispenser specific cash unit with *type billCassette* or the contents of the cash unit are not known this structure will be omitted. If the cash unit is of *type retractCassette* this pointer will be omitted except for the following cases:

- If the retract cash unit is configured to accept level 2 notes then the number and type of level 2 notes is returned in the *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of level 2 notes.
- If items are recognized during retract operations then the number and type of notes

retracted is returned in *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of retracted items.

cashunits/additionalProperties/noteNumberList/noteNumber

Array of banknote numbers the cash unit contains.

cashunits/additionalProperties/noteNumberList/noteNumber/noteID

Identification of note type. The Note ID represents the note identifiers reported by the *CashAcceptor.BanknoteTypes* command. If this value is zero then the note type is unknown.

cashunits/additionalProperties/noteNumberList/noteNumber/count

Actual count of cash items. The value is incremented each time cash items are moved to a cash unit. In the case of recycle cash units this count is decremented as defined in the description of the *logicalCount* field.

cashunits/additionalProperties/noteIDs

Array of integers which contains the note IDs of the banknotes the cash-in cash unit or recycle cash unit can take. This field only applies to *individual* cassette types. If there are no note IDs defined for the cassette or the cassette is not defined as *individual* then *noteIDs* will be omitted.

Event Messages

- [CashManagement.CashUnitErrorEvent](#)
- [CashManagement.NoteErrorEvent](#)
- [CashManagement.CashUnitThresholdEvent](#)
- [CashManagement.CashUnitInfoChangedEvent](#)
- [CashManagement.InfoAvailableEvent](#)
- [CashAcceptor.ShutterStatusChangedEvent](#)

[4.3.9 - CashManagement.EndExchange](#)

This command will end the exchange state. If any physical action took place as a result of the *CashManagement.StartExchange* command then this command will cause the cash units to be returned to their normal physical state, including depositing any remaining items where *exchangeType* is "depositInto". Any necessary device testing will also be initiated.

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

CashManagement.SetCashUnitInfo does not need to be called if the Service can obtain cash unit information from self-configuring cash units.

If an error occurs during the execution of this command, then the client must issue a CashManagement.CashUnitInfo to determine the cash unit information.

A CashManagement.CashUnitErrorEvent will be sent for any cash unit which cannot be successfully updated. If no cash units could be updated then a error code will be returned and CashManagement.CashUnitErrorEvent events generated for every cash unit that could not be updated.

Even if this command does not return a successful completion the exchange state has ended.

Command Message

| Payload | Type | Required |
|--------------------------------------|------|----------|
| { "timeout": 5000 integer } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "cashUnitError" | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

- cashUnitError - A cash unit problem occurred that meant no cash units could be updated. One or more CashManagement.CashUnitErrorEvents will be sent with the details.
- noExchangeActive - There is no exchange active.

Event Messages

- [CashManagement.CashUnitThresholdEvent](#)
- [CashManagement.CashUnitInfoChangedEvent](#)
- [CashManagement.CashUnitErrorEvent](#)
- [CashManagement.NoteErrorEvent](#)
- [CashManagement.InfoAvailableEvent](#)

[4.3.10 - CashManagement.CalibrateCashUnit](#)

This command will cause a vendor dependent sequence of hardware events which will calibrate one cash unit. This is necessary if a new type of bank note is put into the cash unit as the command enables the ATM to obtain the measures of the new bank notes.

This command cannot be used to calibrate cash units which have been locked by the client. A error code will be returned and a CashManagement.CashUnitErrorEvent generated.

Command Message

| Payload | Type | Required |
|---------|------|----------|
|---------|------|----------|

```
{  
    "timeout": 5000,           integer  
    "cashunit": Add example to YAML,  string  
    "numOfBills": 0,           integer  
    "position": {               object  
        "cashunit": Add example to YAML,  string  
        "retractArea": {                 object  
            "outputPosition": "default", string  
            "retractArea": "retract",       string  
            "index": 0                  integer  
        },  
        "outputPosition": "default"     string  
    }  
}
```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

cashunit

The object name of the cash unit as stated by the [CashManagement.GetCashUnitInfo](#) command.

numOfBills

The number of bills to be dispensed during the calibration process.

position

Specifies where the dispensed items should be moved to.

position/cashunit

If defined, this value specifies the object name (as stated by the [CashManagement.GetCashUnitInfo](#) command) of the single cash unit to be used for the storage of any items found.

position/retractArea

This field is used if items are to be moved to internal areas of the device, including cash units, the intermediate stacker, or the transport.

position/retractArea/outputPosition

Output position from which to retract the items. Following values are possible:

- default - The default configuration information should be used.
- left - Retract items from the left output position.
- right - Retract items from the right output position.
- center - Retract items from the center output position.
- top - Retract items from the top output position.
- bottom - Retract items from the bottom output position.
- front - Retract items from the front output position.
- rear - Retract items from the rear output position.

position/retractArea/retractArea

This value specifies the area to which the items are to be retracted. Following values are possible:

- retract - Retract the items to a retract cash unit.
- transport - Retract the items to the transport.
- stacker - Retract the items to the intermediate stacker area.
- reject - Retract the items to a reject cash unit.
- itemCassette - Retract the items to the item cassettes, i.e. cassettes that can be dispensed from.

position/retractArea/index

If *retractArea* is set to "retract" this field defines the position inside the retract cash units

into which the cash is to be retracted. *index* starts with a value of one (1) for the first retract position and increments by one for each subsequent position. If there are several retract cash units (of type "retractCassette" in command CashManagement.CashUnitInfo), *index* would be incremented from the first position of the first retract cash unit to the last position of the last retract cash unit. The maximum value of *index* is the sum of *maximum* of each retract cash unit. If *retractArea* is not set to "retract" the value of this field is ignored.

position/outputPosition

The output position to which items are to be moved. This field is only used if *number* is zero and *retractArea* is omitted. Following values are possible:

- default - The default configuration.
- left - The left output position.
- right - The right output position.
- center - The center output position.
- top - The top output position.
- bottom - The bottom output position.
- front - The front output position.
- rear - The rear output position.

Completion Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "cashUnitError", | string | |
| "cashunit": Add example to YAML, | string | |
| "numOfBills": 0, | integer | |
| "position": { | object | |
| "cashunit": Add example to YAML, | string | |
| "retractArea": { | object | |
| "outputPosition": "default", | string | |
| "retractArea": "retract", | string | |

```
"index": 0           integer  
},  
"outputPosition": "default"      string  
}  
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

- cashUnitError - A cash unit caused an error. A `CashManagement.CashUnitErrorEvent` will be sent with the details.
- unsupportedPosition - The position specified is not valid.
- exchangeActive - The device is in an exchange state.
- invalidCashUnit - The cash unit number specified is not valid.

cashunit

The object name of the cash unit which has been calibrated as stated by the `CashManagement.GetCashUnitInfo` command.

numOfBills

Number of items that were actually dispensed during the calibration process. This value may be different from that passed in using the input structure if the cash dispenser always dispenses a default number of bills. When bills are presented to an output position this is the count of notes presented to the output position, any other notes rejected during the calibration process are not included in this count as they will be accounted for within the

cash unit counts.

position

Specifies where the items were moved to during the calibration process.

position/cashunit

If defined, this value specifies the object name (as stated by the [CashManagement.GetCashUnitInfo](#) command) of the single cash unit to be used for the storage of any items found.

position/retractArea

This field is used if items are to be moved to internal areas of the device, including cash units, the intermediate stacker, or the transport.

position/retractArea/outputPosition

Output position from which to retract the items. Following values are possible:

- default - The default configuration information should be used.
- left - Retract items from the left output position.
- right - Retract items from the right output position.
- center - Retract items from the center output position.
- top - Retract items from the top output position.
- bottom - Retract items from the bottom output position.
- front - Retract items from the front output position.
- rear - Retract items from the rear output position.

position/retractArea/retractArea

This value specifies the area to which the items are to be retracted. Following values are possible:

- retract - Retract the items to a retract cash unit.
- transport - Retract the items to the transport.
- stacker - Retract the items to the intermediate stacker area.
- reject - Retract the items to a reject cash unit.
- itemCassette - Retract the items to the item cassettes, i.e. cassettes that can be dispensed from.

position/retractArea/index

If *retractArea* is set to "retract" this field defines the position inside the retract cash units into which the cash is to be retracted. *index* starts with a value of one (1) for the first retract position and increments by one for each subsequent position. If there are several retract cash units (of type "retractCassette" in command CashManagement.CashUnitInfo), *index* would be incremented from the first position of the first retract cash unit to the last position of the last retract cash unit. The maximum value of *index* is the sum of *maximum* of each retract cash unit. If *retractArea* is not set to "retract" the value of this field is ignored.

position/outputPosition

The output position to which items are to be moved. This field is only used if *number* is zero and *retractArea* is omitted. Following values are possible:

- default - The default configuration.
- left - The left output position.
- right - The right output position.
- center - The center output position.
- top - The top output position.
- bottom - The bottom output position.
- front - The front output position.
- rear - The rear output position.

Event Messages

- [CashManagement.CashUnitThresholdEvent](#)
- [CashManagement.CashUnitInfoChangedEvent](#)
- [CashManagement.CashUnitErrorEvent](#)
- [CashManagement.NoteErrorEvent](#)
- [CashManagement.InfoAvailableEvent](#)
- [Dispenser.ItemsTakenEvent](#)

[4.3.11 - CashManagement.SetClassificationList](#)

This command is used to specify the entire note classification list. Any items not specified in this list will be handled according to normal classification rules. This information is persistent. Information set by this command overrides any existing classification list. If a note is reclassified, it is handled as though it was a note of the new classification. For example, a fit note reclassified as unfit would be treated as though it were unfit, which may mean that the note is not dispensed. Reclassification cannot be used to change a note's classification to a higher level, for example, a note recognized as counterfeit by the device

cannot be reclassified as genuine. In addition, it is not possible to re-classify a level 2 note as level 1. If two or more classification elements specify overlapping note definitions, but different *level* values then the first one takes priority.

Command Message

| Payload | Type | Required |
|--------------------------------------|----------------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "version": Add example to YAML, | string | |
| "classificationElements": [{ | array (object) | |
| "serialNumber": Add example to YAML, | string | |
| "currencyID": Add example to YAML, | string | |
| "value": 0, | number | |
| "level": "level1" | string | |
| }] | | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

version

This is a client defined string that sets the version identifier of the classification list. This property can be omitted if it has no version identifier.

classificationElements

Array of classification objects.

classificationElements/serialNumber

This string defines the serial number or a mask of serial numbers of one element with the defined currency and value. For a definition of the mask see Section Note Classification.

classificationElements/currencyID

The three character ISO 4217 format currency identifier [Ref. 2] of the element.

classificationElements/value

The value of the element. This field can be zero to represent all values.

classificationElements/level

Specifies the note level. Following values are possible:

- level1 - The element specifies notes to be treated as level 1 notes.
- level2 - The element specifies notes to be treated as level 2 notes.
- level3 - The element specifies notes to be treated as level 3 notes.
- level4Fit - The element specifies notes to be treated as fit level 4 notes.
- level4Unfit - The element specifies notes to be treated as unfit level 4 notes.

Completion Message

| <u>Payload</u> | <u>Type</u> | <u>Required</u> |
|---|-------------|-----------------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML | string | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

Event Messages

None

4.4 - Event Messages

4.4.1 - CashManagement.CashUnitErrorEvent

This event is generated if there is a problem with a cash unit during the execution of a command.

| Payload | Type | Required |
|--------------------------------------|---------|----------|
| { | | |
| "failure": "empty", | string | |
| "cashUnit": { | object | |
| "status": "ok", | string | |
| "type": "billCassette", | string | |
| "currencyID": Add example to YAML, | string | |
| "value": 0, | number | |
| "logicalCount": 0, | integer | |
| "maximum": 0, | integer | |
| "appLock": false, | boolean | |
| "cashUnitName": Add example to YAML, | string | |
| "initialCount": 0, | integer | |
| "dispensedCount": 0, | integer | |
| "presentedCount": 0, | integer | |

| | |
|--|-----------------|
| "retractedCount": 0, | integer |
| "rejectCount": 0, | integer |
| "minimum": 0, | integer |
| "physicalPositionName": Add example to YAML, | string |
| "unitID": Add example to YAML, | string |
| "count": 0, | integer |
| "maximumCapacity": 0, | integer |
| "hardwareSensor": false, | boolean |
| "itemType": { | object |
| "all": false, | boolean |
| "unfit": false, | boolean |
| "individual": false, | boolean |
| "level1": false, | boolean |
| "level2": false, | boolean |
| "level3": false, | boolean |
| "itemProcessor": false, | boolean |
| "unfitIndividual": false | boolean |
| } | |
| "cashInCount": 0, | integer |
| "noteNumberList": { | object |
| "noteNumber": [{ | array (object) |
| "noteID": 0, | integer |
| "count": 0 | integer |
| }] | |
| } | |
| "noteIDs": [0] | array (integer) |

}

}

Properties

failure

Specifies the kind of failure that occurred in the cash unit. Following values are possible:

- empty - Specified cash unit is empty.
- error - Specified cash unit has malfunctioned.
- full - Specified cash unit is full.
- locked - Specified cash unit is locked.
- invalid - Specified cash unit is invalid.
- config - An attempt has been made to change the settings of a self-configuring cash unit.
- notConfigured - Specified cash unit is not configured.

cashUnit

The cash unit object that caused the problem.

cashUnit/status

Supplies the status of the cash unit. Following values are possible:

- ok - The cash unit is in a good state.
- full - The cash unit is full.
- high - The cash unit is almost full (i.e. reached or exceeded the threshold defined by *maximum*).
- low - The cash unit is almost empty (i.e. reached or below the threshold defined by *minimum*).
- empty - The cash unit is empty, or insufficient items in the cash unit are preventing further dispense operations.
- inoperative - The cash unit is inoperative.
- missing - The cash unit is missing.
- noValue - The values of the specified cash unit are not available.
- noReference - There is no reference value available for the notes in this cash unit. The cash unit has not been calibrated.
- manipulated - The cash unit has been inserted (including removal followed by a reinsertion) when the device was not in the exchange state. This cash unit cannot be

dispensed from.

cashUnit/type

Type of cash unit. Following values are possible:

- notApplicable - Not applicable. Typically means cash unit is missing.
- rejectCassette - Reject cash unit. This type will also indicate a combined reject/retract cash unit.
- billCassette - Cash unit containing bills.
- coinCylinder - Coin cylinder.
- coinDispenser - Coin dispenser as a whole unit.
- retractCassette - Retract cash unit.
- coupon - Cash unit containing coupons or advertising material.
- document - Cash unit containing documents.
- replenishmentContainer - Replenishment container. A cash unit can be refilled from a replenishment container.
- recycling - Recycling cash unit. This unit is only present when the device implements the Dispenser and CashAcceptor interfaces.
- cashIn - Cash-in cash unit.

cashUnit/currencyID

A three character string storing the ISO format [Ref. 2] Currency ID. This value will be omitted for cash units which contain items of more than one currency type or items to which currency is not applicable. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

cashUnit/value

Supplies the value of a single item in the cash unit. This value is expressed as floating point value. If the *currencyID* field for this cash unit is omitted, then this field will contain zero. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

cashUnit/logicalCount

The meaning of this count depends on the type of cash unit. This value is persistent. For all cash units except retract cash units (*type* is not *retractCassette*) this value specifies the number of items inside the cash unit. For all dispensing cash units (*type* is *billCassette*, *coinCylinder*, *coinDispenser*, *coupon*, *document* or *recycling*), this value includes any items from the cash unit not yet presented to the customer. This count is only decremented when the items are either known to be in customer access or successfully rejected. If the cash

unit is usable from the CashAcceptor interface (*type* is *recycling*, *cashIn*, *retractCassette* or *rejectCassette*) then this value will be incremented as a result of a cash-in operation. Note that for a reject cash unit (*type* is *rejectCassette*), this value is unreliable, since the typical reason for dumping items to the reject cash unit is a suspected count failure. For a retract cash unit (*type* is *retractCassette*) this value specifies the number of retract operations which result in items entering the cash unit.

cashUnit/maximum

When *count* reaches this value the threshold event

CashManagement.CashUnitThresholdEvent (*high*) will be generated. This value can be different from the actual capacity of the cassette. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

cashUnit/appLock

If this value is TRUE items cannot be dispensed from or deposited into the cash unit. If this value is TRUE and the client attempts to use the cash unit a CashManagement.CashUnitErrorEvent event will be generated and an error completion message will be returned. This value is persistent.

cashUnit/cashUnitName

A name which helps to identify the type of the cash unit. This is especially useful in the case of cash units of type *document* where different documents can have the same currency and value. For example, travelers checks and bank checks may have the same currency and value but still need to be identifiable as different types of document. Where this value is not relevant (e.g. in bill cash units) the property can be omitted. This value is persistent.

cashUnit/initialCount

Initial number of items contained in the cash unit. This value is persistent.

cashUnit/dispensedCount

The number of items dispensed from this cash unit. This count is incremented when the items are removed from the cash units. This count includes any items that were rejected during the dispense operation and are no longer in this cash unit. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

cashUnit/presentedCount

The number of items from this cash unit that have been presented to the customer. This count is incremented when the items are presented to the customer. If it is unknown if a customer has been presented with the items, then this count is not updated. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

cashUnit/retractedCount

The number of items that have been accessible to a customer and retracted into the cash unit. This value is persistent.

cashUnit/rejectCount

The number of items dispensed from this cash unit which have been rejected, are in a cash unit other than this cash unit, and which have not been accessible to a customer. This value may be unreliable, since a typical reason for rejecting items is a suspected pick failure. Other reasons for rejecting items may include incorrect note denominations, classifications not valid for dispensing, or where the transaction has been cancelled and a Reject command has been called. For reject and retract cash units (*type* is *rejectCassette* or *retractCassette*) this field does not apply and will be reported as zero. This value is persistent.

cashUnit/minimum

This field is not applicable to retract and reject cash units. For all cash units which dispense items (all other), when *count* reaches this value the threshold event *CashManagement.CashUnitThresholdEvent (low)* will be generated. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

cashUnit/physicalPositionName

A name identifying the physical location of the cash unit.

cashUnit/unitID

A 5 character string uniquely identifying the cash unit.

cashUnit/count

As defined by the *logicalCount* description, but with the following exceptions: This count does not include items dispensed but not yet presented. On cash units with *type* set to "retractCassette" the count represents the number of items, unless the device cannot count items during a retract, in which case this count will be zero. This value is persistent.

cashUnit/maximumCapacity

The maximum number of items the cash unit can hold. This is only for informational purposes. No threshold event CashManagement.CashUnitThresholdEvent will be generated. This value is persistent.

cashUnit/hardwareSensor

Specifies whether or not threshold events can be generated based on hardware sensors in the device. If this value is TRUE then threshold events may be generated based on hardware sensors as opposed to counts.

cashUnit/itemType

Specifies the type of items the cash unit takes as a combination of the following flags. The table in the Comments section of this command defines how to interpret the combination of these flags (TODO: include Table)

cashUnit/itemType/all

The cash unit takes all fit banknote types. These are level 4 notes which are fit for recycling.

cashUnit/itemType/unfit

The cash unit takes all unfit banknotes. These are level 4 notes which are unfit for recycling.

cashUnit/itemType/individual

The cash unit takes all types of fit banknotes specified in an individual list. These are level 4 notes which are fit for recycling.

cashUnit/itemType/level1

Level 1 note types are stored in this cash unit.

cashUnit/itemType/level2

If notes can be classified as level 2, then level 2 note types are stored in this cash unit.

cashUnit/itemType/level3

If notes can be classified as level 3, then level 3 note types are stored in this cash unit.

cashUnit/itemType/itemProcessor

The cash unit can accept items on the ItemProcessor interface.

cashUnit/itemType/unfitIndividual

The cash unit takes all types of unfit banknotes specified in an individual list. These are level 4 notes which are unfit for recycling.

cashUnit/cashInCount

Count of items that have entered the cash unit. This counter is incremented whenever an item enters a cash unit for any reason, unless it originated from this cash unit but was returned without being accessible to a customer. For a retract cash unit this value represents the total number of items of all types in the cash unit, or if the device cannot count items during a retract operation this value will be zero. This value is persistent.

cashUnit/noteNumberList

Array of cash items inside the cash unit. The content of this structure is persistent. If the cash unit is Dispenser specific cash unit with *type billCassette* or the contents of the cash unit are not known this structure will be omitted. If the cash unit is of *type retractCassette* this pointer will be omitted except for the following cases:

- If the retract cash unit is configured to accept level 2 notes then the number and type of level 2 notes is returned in the *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of level 2 notes.
- If items are recognized during retract operations then the number and type of notes retracted is returned in *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of retracted items.

cashUnit/noteNumberList/noteNumber

Array of banknote numbers the cash unit contains.

cashUnit/noteNumberList/noteNumber/noteID

Identification of note type. The Note ID represents the note identifiers reported by the *CashAcceptor.BanknoteTypes* command. If this value is zero then the note type is unknown.

cashUnit/noteNumberList/noteNumber/count

Actual count of cash items. The value is incremented each time cash items are moved to a cash unit. In the case of recycle cash units this count is decremented as defined in the description of the *logicalCount* field.

cashUnit/noteIDs

Array of integers which contains the note IDs of the banknotes the cash-in cash unit or recycle cash unit can take. This field only applies to *individual* cassette types. If there are no note IDs defined for the cassette or the cassette is not defined as *individual* then *noteIDs* will be omitted.

[4.4.2 - CashManagement.NoteErrorEvent](#)

This event specifies the reason for a note detection error during the execution of a command.

| Payload | Type | Required |
|------------------------|--------|----------|
| { | | |
| "reason": "doubleNote" | string | |
| } | | |

Properties**reason**

The reason for the notes detection error. Following values are possible:

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

- doubleNote - Double notes have been detected.
- longNote - A long note has been detected.
- skewedNote - A skewed note has been detected.
- incorrectCount - An item counting error has occurred.
- notesTooClose - Notes have been detected as being too close.
- otherNoteError - An item error not covered by the other values has been detected.
- shortNote - Short notes have been detected.

4.4.3 - CashManagement.InfoAvailableEvent

This execute event is generated when information is available for items detected during the cash processing operation.

| Payload | Type | Required |
|-----------------------|----------------|----------|
| { | | |
| "itemInfoSummary": [{ | array (object) | |
| "level": "level1", | string | |
| "numOfItems": 0 | integer | |
| }] | | |
| } | | |

Properties

itemInfoSummary

Array of itemInfoSummary objects, one object for every level.

itemInfoSummary/level

Defines the note level. Following values are possible:

- level1 - Information for level 1 notes.
- level2 - Information for level 2 notes.
- level3 - Information for level 3 notes.
- level4 - Information for level 4 notes.

itemInfoSummary/numOfItems

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

Number of items classified as *level* which have information available.

[4.4.4 - CashAcceptor.ShutterStatusChangedEvent](#)

Within the limitations of the hardware sensors this service event is generated whenever the status of a shutter changes. The shutter status can change because of an explicit, implicit or manual operation depending on how the shutter is operated.

| Payload | Type | Required |
|-----------------------|--------|----------|
| { | | |
| "position": "inLeft", | string | |
| "shutter": "closed" | string | |
| } | | |

Properties

position

Specifies one of the input or output positions whose shutter status has changed. Following values are possible:

"inLeft": Left input position.

"inRight": Right input position.

"inCenter": Center input position.

"inTop": Top input position.

"inBottom": Bottom input position.

"inFront": Front input position.

"inRear": Rear input position.

"outLeft": Left output position.

"outRight": Right output position.

"outCenter": Center output position.

"outTop": Top output position.

"outBottom": Bottom output position.

"outFront": Front output position.

"outRear": Rear output position.

shutter

Specifies the new state of the shutter. Following values are possible:

"closed": The shutter is closed.

"open": The shutter is opened.

"jammed": The shutter is jammed.

"unknown": Due to a hardware error or other condition, the state of the shutter cannot be determined.

[4.4.5 - Dispenser.ItemsTakenEvent](#)

This event is generated when items presented to the user have been taken. This event may be generated at any time.

| Payload | Type | Required |
|-----------------------|--------|----------|
| { | | |
| "position": "default" | string | |
| } | | |

Properties

position

The output position from which the items have been removed. Following values are possible:

- default - The default configuration.
- left - The left output position.
- right - The right output position.
- center - The center output position.
- top - The top output position.

- bottom - The bottom output position.
- front - The front output position.
- rear - The rear output position.

4.5 - Unsolicited Messages

4.5.1 - [CashManagement.SafeDoorOpenEvent](#)

This event specifies that the safe door has been opened.

4.5.2 - [CashManagement.SafeDoorClosedEvent](#)

This event specifies that the safe door has been closed.

4.5.3 - [CashManagement.CashUnitInfoChangedEvent](#)

This service event is generated under the following circumstances:

- It is generated whenever `status` changes. For instance, a cash unit has been removed or inserted, or a cash unit has become empty or full.
- This event will also be generated for every cash unit changed in any way (including changes to counts, e.g. `count`, `rejectCount`, `initialCount`, `dispensedCount` and `presentedCount`) as a result of the [CashManagement.SetCashUnitInfo](#) command.
- This event will also be fired when any change is made to a cash unit by the following commands, except for changes to counts (e.g. `count`, `rejectCount`, `initialCount`, `dispensedCount` and `presentedCount`):

[Dispenser.CalibrateCashUnit](#)

[Dispenser.TestCashUnit](#)

- In addition this event will be generated when a cash unit has been counted during the [CashAcceptor.CashUnitCount](#) command execution.

When a cash unit is removed, the status of the cash unit becomes missing.

If a new cash unit is inserted the cash unit structure reported by the last [CashManagement.GetCashUnitInfo](#) command is no longer valid. In that case a client should issue a [CashManagement.GetCashUnitInfo](#) command after receiving this event to obtain updated cash unit information.

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "status": "ok", | string | |
| "type": "billCassette", | string | |
| "currencyID": Add example to YAML, | string | |
| "value": 0, | number | |
| "logicalCount": 0, | integer | |
| "maximum": 0, | integer | |
| "appLock": false, | boolean | |
| "cashUnitName": Add example to YAML, | string | |
| "initialCount": 0, | integer | |
| "dispensedCount": 0, | integer | |
| "presentedCount": 0, | integer | |
| "retractedCount": 0, | integer | |
| "rejectCount": 0, | integer | |
| "minimum": 0, | integer | |
| "physicalPositionName": Add example to YAML, | string | |
| "unitID": Add example to YAML, | string | |
| "count": 0, | integer | |
| "maximumCapacity": 0, | integer | |
| "hardwareSensor": false, | boolean | |
| "itemType": { | object | |
| "all": false, | boolean | |
| "unfit": false, | boolean | |
| "individual": false, | boolean | |
| "level1": false, | boolean | |
| "level2": false, | boolean | |

```

"level3": false,                                boolean
"itemProcessor": false,                           boolean
"unfitIndividual": false,                        boolean
},
"cashInCount": 0,                               integer
"noteNumberList": {                             object
  "noteNumber": [{                            array (object)
    "noteID": 0,                                integer
    "count": 0                                   integer
  }]
},
"noteIDs": [0]                                    array (integer)
}

```

Properties

status

Supplies the status of the cash unit. Following values are possible:

- ok - The cash unit is in a good state.
- full - The cash unit is full.
- high - The cash unit is almost full (i.e. reached or exceeded the threshold defined by *maximum*).
- low - The cash unit is almost empty (i.e. reached or below the threshold defined by *minimum*).
- empty - The cash unit is empty, or insufficient items in the cash unit are preventing further dispense operations.
- inoperative - The cash unit is inoperative.
- missing - The cash unit is missing.
- noValue - The values of the specified cash unit are not available.
- noReference - There is no reference value available for the notes in this cash unit. The cash unit has not been calibrated.
- manipulated - The cash unit has been inserted (including removal followed by a reinsertion) when the device was not in the exchange state. This cash unit cannot be

dispensed from.

type

Type of cash unit. Following values are possible:

- notApplicable - Not applicable. Typically means cash unit is missing.
- rejectCassette - Reject cash unit. This type will also indicate a combined reject/retract cash unit.
- billCassette - Cash unit containing bills.
- coinCylinder - Coin cylinder.
- coinDispenser - Coin dispenser as a whole unit.
- retractCassette - Retract cash unit.
- coupon - Cash unit containing coupons or advertising material.
- document - Cash unit containing documents.
- replenishmentContainer - Replenishment container. A cash unit can be refilled from a replenishment container.
- recycling - Recycling cash unit. This unit is only present when the device implements the Dispenser and CashAcceptor interfaces.
- cashIn - Cash-in cash unit.

currencyID

A three character string storing the ISO format [Ref. 2] Currency ID. This value will be omitted for cash units which contain items of more than one currency type or items to which currency is not applicable. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

value

Supplies the value of a single item in the cash unit. This value is expressed as floating point value. If the *currencyID* field for this cash unit is omitted, then this field will contain zero. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

logicalCount

The meaning of this count depends on the type of cash unit. This value is persistent. For all cash units except retract cash units (*type* is not *retractCassette*) this value specifies the number of items inside the cash unit. For all dispensing cash units (*type* is *billCassette*, *coinCylinder*, *coinDispenser*, *coupon*, *document* or *recycling*), this value includes any items from the cash unit not yet presented to the customer. This count is only decremented when the items are either known to be in customer access or successfully rejected. If the cash

unit is usable from the CashAcceptor interface (*type* is *recycling*, *cashIn*, *retractCassette* or *rejectCassette*) then this value will be incremented as a result of a cash-in operation. Note that for a reject cash unit (*type* is *rejectCassette*), this value is unreliable, since the typical reason for dumping items to the reject cash unit is a suspected count failure. For a retract cash unit (*type* is *retractCassette*) this value specifies the number of retract operations which result in items entering the cash unit.

maximum

When *count* reaches this value the threshold event

CashManagement.CashUnitThresholdEvent (*high*) will be generated. This value can be different from the actual capacity of the cassette. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

appLock

If this value is TRUE items cannot be dispensed from or deposited into the cash unit. If this value is TRUE and the client attempts to use the cash unit a CashManagement.CashUnitErrorEvent event will be generated and an error completion message will be returned. This value is persistent.

cashUnitName

A name which helps to identify the type of the cash unit. This is especially useful in the case of cash units of type *document* where different documents can have the same currency and value. For example, travelers checks and bank checks may have the same currency and value but still need to be identifiable as different types of document. Where this value is not relevant (e.g. in bill cash units) the property can be omitted. This value is persistent.

initialCount

Initial number of items contained in the cash unit. This value is persistent.

dispensedCount

The number of items dispensed from this cash unit. This count is incremented when the items are removed from the cash units. This count includes any items that were rejected during the dispense operation and are no longer in this cash unit. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

presentedCount

The number of items from this cash unit that have been presented to the customer. This count is incremented when the items are presented to the customer. If it is unknown if a customer has been presented with the items, then this count is not updated. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

retractedCount

The number of items that have been accessible to a customer and retracted into the cash unit. This value is persistent.

rejectCount

The number of items dispensed from this cash unit which have been rejected, are in a cash unit other than this cash unit, and which have not been accessible to a customer. This value may be unreliable, since a typical reason for rejecting items is a suspected pick failure. Other reasons for rejecting items may include incorrect note denominations, classifications not valid for dispensing, or where the transaction has been cancelled and a Reject command has been called. For reject and retract cash units (*type* is *rejectCassette* or *retractCassette*) this field does not apply and will be reported as zero. This value is persistent.

minimum

This field is not applicable to retract and reject cash units. For all cash units which dispense items (all other), when *count* reaches this value the threshold event *CashManagement.CashUnitThresholdEvent (low)* will be generated. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

physicalPositionName

A name identifying the physical location of the cash unit.

unitID

A 5 character string uniquely identifying the cash unit.

count

As defined by the *logicalCount* description, but with the following exceptions: This count does not include items dispensed but not yet presented. On cash units with *type* set to "retractCassette" the count represents the number of items, unless the device cannot count items during a retract, in which case this count will be zero. This value is persistent.

maximumCapacity

The maximum number of items the cash unit can hold. This is only for informational purposes. No threshold event CashManagement.CashUnitThresholdEvent will be generated. This value is persistent.

hardwareSensor

Specifies whether or not threshold events can be generated based on hardware sensors in the device. If this value is TRUE then threshold events may be generated based on hardware sensors as opposed to counts.

itemType

Specifies the type of items the cash unit takes as a combination of the following flags. The table in the Comments section of this command defines how to interpret the combination of these flags (TODO: include Table)

itemType/all

The cash unit takes all fit banknote types. These are level 4 notes which are fit for recycling.

itemType/unfit

The cash unit takes all unfit banknotes. These are level 4 notes which are unfit for recycling.

itemType/individual

The cash unit takes all types of fit banknotes specified in an individual list. These are level 4 notes which are fit for recycling.

itemType/level1

Level 1 note types are stored in this cash unit.

itemType/level2

If notes can be classified as level 2, then level 2 note types are stored in this cash unit.

itemType/level3

If notes can be classified as level 3, then level 3 note types are stored in this cash unit.

itemType/itemProcessor

The cash unit can accept items on the ItemProcessor interface.

itemType/unfitIndividual

The cash unit takes all types of unfit banknotes specified in an individual list. These are level 4 notes which are unfit for recycling.

cashInCount

Count of items that have entered the cash unit. This counter is incremented whenever an item enters a cash unit for any reason, unless it originated from this cash unit but was returned without being accessible to a customer. For a retract cash unit this value represents the total number of items of all types in the cash unit, or if the device cannot count items during a retract operation this value will be zero. This value is persistent.

noteNumberList

Array of cash items inside the cash unit. The content of this structure is persistent. If the cash unit is Dispenser specific cash unit with *type billCassette* or the contents of the cash unit are not known this structure will be omitted. If the cash unit is of *type retractCassette* this pointer will be omitted except for the following cases:

- If the retract cash unit is configured to accept level 2 notes then the number and type of level 2 notes is returned in the *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of level 2 notes.
- If items are recognized during retract operations then the number and type of notes retracted is returned in *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of retracted items.

noteNumberList/noteNumber

Array of banknote numbers the cash unit contains.

noteNumberList/noteNumber/noteID

Identification of note type. The Note ID represents the note identifiers reported by the *CashAcceptor.BanknoteTypes* command. If this value is zero then the note type is unknown.

noteNumberList/noteNumber/count

Actual count of cash items. The value is incremented each time cash items are moved to a cash unit. In the case of recycle cash units this count is decremented as defined in the description of the *logicalCount* field.

noteIDs

Array of integers which contains the note IDs of the banknotes the cash-in cash unit or recycle cash unit can take. This field only applies to *individual* cassette types. If there are no note IDs defined for the cassette or the cassette is not defined as *individual* then *noteIDs* will be omitted.

4.5.4 - CashManagement.TellerInfoChangedEvent

This service event is generated when the counts assigned to a teller have changed. This event is only returned as a result of a CashManagement.SetTellerInfo command.

| Payload | Type | Required |
|---------------|---------|----------|
| { | | |
| "tellerID": 0 | integer | |
| } | | |

Properties**tellerID**

Integer holding the ID of the teller whose counts have changed.

4.5.5 - CashManagement.CashUnitThresholdEvent

This user event is generated when a threshold condition has occurred in one of the cash units.

This event can be triggered either by hardware sensors in the device or by the **count** reaching the **minimum** or **maximum** value as specified in the GetCashUnitInfo structure.

The client can check if the device has hardware sensors by querying the **hardwareSensor** field of the cash unit structure. If a cash unit has this capability then threshold events based on hardware sensors will be triggered if the **maximum** or **minimum** values are not used or are set to zero.

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "status": "ok", | string | |
| "type": "billCassette", | string | |
| "currencyID": Add example to YAML, | string | |
| "value": 0, | number | |
| "logicalCount": 0, | integer | |
| "maximum": 0, | integer | |
| "appLock": false, | boolean | |
| "cashUnitName": Add example to YAML, | string | |
| "initialCount": 0, | integer | |
| "dispensedCount": 0, | integer | |
| "presentedCount": 0, | integer | |
| "retractedCount": 0, | integer | |
| "rejectCount": 0, | integer | |
| "minimum": 0, | integer | |
| "physicalPositionName": Add example to YAML, | string | |
| "unitID": Add example to YAML, | string | |
| "count": 0, | integer | |

```

"maximumCapacity": 0,           integer
"hardwareSensor": false,        boolean
"itemType": {                   object
  "all": false,                boolean
  "unfit": false,              boolean
  "individual": false,         boolean
  "level1": false,              boolean
  "level2": false,              boolean
  "level3": false,              boolean
  "itemProcessor": false,       boolean
  "unfitIndividual": false    boolean
},
"cashInCount": 0,               integer
"noteNumberList": {             object
  "noteNumber": [{              array (object)
    "noteID": 0,                 integer
    "count": 0                  integer
  }]
},
"noteIDs": [0]                  array (integer)
}

```

Properties

status

Supplies the status of the cash unit. Following values are possible:

- ok - The cash unit is in a good state.
- full - The cash unit is full.

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

- high - The cash unit is almost full (i.e. reached or exceeded the threshold defined by *maximum*).
- low - The cash unit is almost empty (i.e. reached or below the threshold defined by *minimum*).
- empty - The cash unit is empty, or insufficient items in the cash unit are preventing further dispense operations.
- inoperative - The cash unit is inoperative.
- missing - The cash unit is missing.
- noValue - The values of the specified cash unit are not available.
- noReference - There is no reference value available for the notes in this cash unit. The cash unit has not been calibrated.
- manipulated - The cash unit has been inserted (including removal followed by a reinsertion) when the device was not in the exchange state. This cash unit cannot be dispensed from.

type

Type of cash unit. Following values are possible:

- notApplicable - Not applicable. Typically means cash unit is missing.
- rejectCassette - Reject cash unit. This type will also indicate a combined reject/retract cash unit.
- billCassette - Cash unit containing bills.
- coinCylinder - Coin cylinder.
- coinDispenser - Coin dispenser as a whole unit.
- retractCassette - Retract cash unit.
- coupon - Cash unit containing coupons or advertising material.
- document - Cash unit containing documents.
- replenishmentContainer - Replenishment container. A cash unit can be refilled from a replenishment container.
- recycling - Recycling cash unit. This unit is only present when the device implements the Dispenser and CashAcceptor interfaces.
- cashIn - Cash-in cash unit.

currencyID

A three character string storing the ISO format [Ref. 2] Currency ID. This value will be omitted for cash units which contain items of more than one currency type or items to which currency is not applicable. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

value

Supplies the value of a single item in the cash unit. This value is expressed as floating point value. If the *currencyID* field for this cash unit is omitted, then this field will contain zero. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

logicalCount

The meaning of this count depends on the type of cash unit. This value is persistent. For all cash units except retract cash units (*type* is not *retractCassette*) this value specifies the number of items inside the cash unit. For all dispensing cash units (*type* is *billCassette*, *coinCylinder*, *coinDispenser*, *coupon*, *document* or *recycling*), this value includes any items from the cash unit not yet presented to the customer. This count is only decremented when the items are either known to be in customer access or successfully rejected. If the cash unit is usable from the CashAcceptor interface (*type* is *recycling*, *cashIn*, *retractCassette* or *rejectCassette*) then this value will be incremented as a result of a cash-in operation. Note that for a reject cash unit (*type* is *rejectCassette*), this value is unreliable, since the typical reason for dumping items to the reject cash unit is a suspected count failure. For a retract cash unit (*type* is *retractCassette*) this value specifies the number of retract operations which result in items entering the cash unit.

maximum

When *count* reaches this value the threshold event

CashManagement.CashUnitThresholdEvent (high) will be generated. This value can be different from the actual capacity of the cassette. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

appLock

If this value is TRUE items cannot be dispensed from or deposited into the cash unit. If this value is TRUE and the client attempts to use the cash unit a *CashManagement.CashUnitErrorEvent* event will be generated and an error completion message will be returned. This value is persistent.

cashUnitName

A name which helps to identify the type of the cash unit. This is especially useful in the case of cash units of type *document* where different documents can have the same currency and value. For example, travelers checks and bank checks may have the same currency and value but still need to be identifiable as different types of document. Where this value is not relevant (e.g. in bill cash units) the property can be omitted. This value is persistent.

initialCount

Initial number of items contained in the cash unit. This value is persistent.

dispensedCount

The number of items dispensed from this cash unit. This count is incremented when the items are removed from the cash units. This count includes any items that were rejected during the dispense operation and are no longer in this cash unit. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

presentedCount

The number of items from this cash unit that have been presented to the customer. This count is incremented when the items are presented to the customer. If it is unknown if a customer has been presented with the items, then this count is not updated. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

retractedCount

The number of items that have been accessible to a customer and retracted into the cash unit. This value is persistent.

rejectCount

The number of items dispensed from this cash unit which have been rejected, are in a cash unit other than this cash unit, and which have not been accessible to a customer. This value may be unreliable, since a typical reason for rejecting items is a suspected pick failure. Other reasons for rejecting items may include incorrect note denominations, classifications not valid for dispensing, or where the transaction has been cancelled and a Reject command has been called. For reject and retract cash units (*type* is *rejectCassette* or *retractCassette*) this field does not apply and will be reported as zero. This value is persistent.

minimum

This field is not applicable to retract and reject cash units. For all cash units which dispense items (all other), when *count* reaches this value the threshold event CashManagement.CashUnitThresholdEvent (*low*) will be generated. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero

then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

physicalPositionName

A name identifying the physical location of the cash unit.

unitID

A 5 character string uniquely identifying the cash unit.

count

As defined by the *logicalCount* description, but with the following exceptions: This count does not include items dispensed but not yet presented. On cash units with *type* set to "retractCassette" the count represents the number of items, unless the device cannot count items during a retract, in which case this count will be zero. This value is persistent.

maximumCapacity

The maximum number of items the cash unit can hold. This is only for informational purposes. No threshold event CashManagement.CashUnitThresholdEvent will be generated. This value is persistent.

hardwareSensor

Specifies whether or not threshold events can be generated based on hardware sensors in the device. If this value is TRUE then threshold events may be generated based on hardware sensors as opposed to counts.

itemType

Specifies the type of items the cash unit takes as a combination of the following flags. The table in the Comments section of this command defines how to interpret the combination of these flags (TODO: include Table)

itemType/all

The cash unit takes all fit banknote types. These are level 4 notes which are fit for recycling.

itemType/unfit

The cash unit takes all unfit banknotes. These are level 4 notes which are unfit for recycling.

itemType/individual

The cash unit takes all types of fit banknotes specified in an individual list. These are level 4 notes which are fit for recycling.

itemType/level1

Level 1 note types are stored in this cash unit.

itemType/level2

If notes can be classified as level 2, then level 2 note types are stored in this cash unit.

itemType/level3

If notes can be classified as level 3, then level 3 note types are stored in this cash unit.

itemType/itemProcessor

The cash unit can accept items on the ItemProcessor interface.

itemType/unfitIndividual

The cash unit takes all types of unfit banknotes specified in an individual list. These are level 4 notes which are unfit for recycling.

cashInCount

Count of items that have entered the cash unit. This counter is incremented whenever an item enters a cash unit for any reason, unless it originated from this cash unit but was returned without being accessible to a customer. For a retract cash unit this value represents the total number of items of all types in the cash unit, or if the device cannot count items during a retract operation this value will be zero. This value is persistent.

noteNumberList

Array of cash items inside the cash unit. The content of this structure is persistent. If the cash unit is Dispenser specific cash unit with *type billCassette* or the contents of the cash unit are not known this structure will be omitted. If the cash unit is of *type retractCassette* this pointer will be omitted except for the following cases:

- If the retract cash unit is configured to accept level 2 notes then the number and type of level 2 notes is returned in the *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of level 2 notes.
- If items are recognized during retract operations then the number and type of notes retracted is returned in *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of retracted items.

noteNumberList/noteNumber

Array of banknote numbers the cash unit contains.

noteNumberList/noteNumber/noteID

Identification of note type. The Note ID represents the note identifiers reported by the *CashAcceptor.BanknoteTypes* command. If this value is zero then the note type is unknown.

noteNumberList/noteNumber/count

Actual count of cash items. The value is incremented each time cash items are moved to a cash unit. In the case of recycle cash units this count is decremented as defined in the description of the *logicalCount* field.

noteIDs

Array of integers which contains the note IDs of the banknotes the cash-in cash unit or recycle cash unit can take. This field only applies to *individual* cassette types. If there are no note IDs defined for the cassette or the cassette is not defined as *individual* then *noteIDs* will be omitted.

5 - Dispenser Interface

This chapter defines the Dispenser interface functionality and messages.

5.1 - Summary

This specification describes the functionality of an XFS4IoT compliant Cash Dispenser interface. It defines the service-specific commands that can be issued to the service using the WebSocket endpoint.

Persistent values are maintained through power failures, open sessions, close session and system resets.

This specification covers the dispensing of items. An "item" is defined as any media that can be dispensed and includes coupons, documents, bills and coins.

5.2 - Command Messages

5.2.1 - Dispenser.GetMixTypes

This command is used to obtain a list of supported mix algorithms and available house mix tables.

Command Message

| Payload | Type | Required |
|-----------------|---------|----------|
| { | | |
| "timeout": 5000 | integer | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "mixTypes": [{ | array (object) | |
| "mixNumber": 0, | integer | |
| "mixType": "mixAlgorithm", | string | |
| "subType": 0, | integer | |
| "name": Add example to YAML | string | |
| }] | | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

mixTypes

Array of mix type objects.

mixTypes/mixNumber

Number identifying the mix algorithm or the house mix table. This number can be passed

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

to the Dispenser.MixTable, Dispenser.Dispense and Dispenser.Denominate commands.

`mixTypes/mixType`

Specifies whether the mix type is an algorithm or a house mix table. Possible values are `mixAlgorithm` and `mixTable`.

`mixTypes/subType`

Contains a vendor-defined number that identifies the type of algorithm. Individual vendor-defined mix algorithms are defined above hexadecimal 7FFF. Mix algorithms which are provided by the Service are in the range hexadecimal 8000 - 8FFF. Client defined mix algorithms start at hexadecimal 9000. All numbers below 8000 hexadecimal are reserved. If `mixType` is "mixTable", this value will be zero.

`mixTypes/name`

Name of the table/algorithm used.

Event Messages

None

5.2.2 - Dispenser.GetMixTable

This command is used to obtain the house mix table specified by the supplied mix number.

Command Message

| Payload | Type | Required |
|-------------------------------|---------|----------|
| { | | |
| <code>"timeout": 5000,</code> | integer | |
| <code>"mixNumber": 0</code> | integer | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

mixNumber

Number of the requested house mix table.

Completion Message

| Payload | Type | Required |
|--|-----------------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "invalidMixNumber", | string | |
| "mixNumber": 0, | integer | |
| "name": Add example to YAML, | string | |
| "mixHeader": [0], | array (number) | |
| "mixRows": [{ | array (object) | |
| "amount": 0, | number | |
| "mixture": [0] | array (integer) | |
| }] | | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

- invalidMixNumber - The *mixNumber* parameter does not correspond to a defined mix table.

mixNumber

Number identifying the house mix table.

name

Name of the house mix table.

mixHeader

Array of floating point numbers; each element defines the value of the item corresponding to its respective column.

mixRows

Array of rows of the mix table.

mixRows/amount

Amount denominated by this mix row.

mixRows/mixture

A mix row, an array of integers; each element defines the quantity of each item denomination in the mix used in the denomination of *amount*. The value of each array element is defined by the *mixHeader*.

Event Messages

None

5.2.3 - Dispenser.GetPresentStatus

This command is used to obtain the status of the most recent attempt to dispense and/or present items to the customer from a specified output position. The items may have been dispensed and/or presented as a result of the Dispenser.Present or Dispenser.Dispense command. This status is not updated as a result of any other command that can dispense/present items.

This value is persistent and is valid until the next time an attempt is made to present or dispense items to the customer.

The denominations reported by this command may not accurately reflect the operation if the cash units have been re-configured (e.g. if the values associated with a cash unit are changed, or new cash units are configured).

Command Message

| Payload | Type | Required |
|---|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "position": "default", | string | |
| "nonce": 646169ECDD0E440C2CECC8DDD7C27C22 | string | |
| } | | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

position

Required output position. Following values are possible:

- default - The default configuration.
- left - The left output position.
- right - The right output position.
- center - The center output position.
- top - The top output position.
- bottom - The bottom output position.
- front - The front output position.
- rear - The rear output position.

nonce

A nonce value to be used when creating the end to end security token in the response. See the API documentation on end to end security for more details.

Completion Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "unsupportedPosition", | string | |
| "denomination": { | object | |
| "currencies": { | object | |
| "additionalProperties": 0 | number | |
| }, | | |
| "values": { | object | |
| "additionalProperties": 0 | integer | |
| }, | | |
| "cashBox": 0 | integer | |
| }, | | |
| "presentState": "presented", | string | |

```
"extra": [Add example to YAML],           array (string)
"token": Add example to YAML             string
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

- unsupportedPosition - The specified output position is not supported.

denomination

Denomination structure which contains the amount dispensed from the specified output position and the number of items dispensed from each cash unit. Where the capability *moveItems* reports *toStacker* this value is cumulative across a series of Dispenser.Dispense calls that add additional items to the stacker. Where mixed currencies were dispensed the *amount* field in the returned denomination structure will be zero and the *currencyID* field will be omitted.

denomination/currencies

"List of currency and amount combinations for denomination. There will be one entry for each currency in the denomination. The property name is the currency name in ISO format (e.g. "EUR").

denomination/currencies/additionalProperties

The amount to be denominated or dispensed. Use currency identifier in ISO format as

property name.

denomination/values

This list specifies the number of items to take from the cash units. Each entry uses a cashunit object name as stated by the [CashManagement.GetCashUnitInfo](#) command. The value of the entry is the number of items to take from that unit. If the client does not wish to specify a denomination, it should omit the values property.

denomination/values/additionalProperties

Number of items to take from the specified cash unit. Use cash unit name as specified by the [CashManagement.GetCashUnitInfo](#) command as property name.

denomination/cashBox

Only applies to Teller Dispensers. Amount to be paid from the teller's cash box.

presentState

Supplies the status of the last dispense or present operation. Following values are possible:

- presented - The items were presented. This status is set as soon as the customer has access to the items.
- notPresented - The customer has not had access to the items.
- unknown - It is not known if the customer had access to the items.

extra

Pointer to a list of vendor-specific, or any other extended, information. The information is returned as a series of "key=value" strings so that it is easily extensible by Service Providers.

token

The token that validates the present status and guarantees that the details have not been tampered with.

This token will follow the standard token format, defined in the API documentation, and will contain the following key:

- DISPENSEID: HEX encoded HMACSHA256 value from the token for the last dispense operation. This is included so that the present status can be linked to a previously

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

authorised dispense. If this doesn't match the expected value then the receiver of the token may assume that the transaction is suspect, and that the cash may have been accessible to the customer. If there was no dispense token this key will not be included - This may also mark the dispense as suspect.

- DISPENSED1: The total value of a single currency that was removed from cassettes and possibly stacked inside the machine ready to present. This does not include any notes moved to the reject cassette. This will be a number string that may contain a fractional part. The decimal character will be ". ". The value, including the fractional part, will be defined by the ISO currency. The number will be followed by the ISO currency code. The currency code will be upper case.

For example, "123.45EUR" will be €123 and 45 cents.

- PRESENTED1: May be "YES" or "NO" (upper case.) This will be YES if the notes could at any time have been accessible outside the machine and may have been tampered with. If the notes were never accessible outside the machine and can not have been tampered with then this value will be NO.
- PRESENTEDAMOUNT1: The total value of a single currency that was presented outside of the machine. The format is the same as for DISPENSED1.
- RETRACTED1: May be "YES" or "NO". If notes were accessible outside of the machine and were then retracted back into the machine then this value will be YES. If notes were never presented, or all notes that were presented were not retracted, then this will be NO.
- RETRACTEDAMOUNT1: If notes are counted during a retract this will be the amount retracted. If the notes aren't counted, or the value of the notes retracted is not reliably known for any reason, this will be "?"

For example:

"RETRACTEDAMOUNT1=123.45EUR" €123 and 45 cents was retracted and counted.

"RETRACTEDAMOUNT1=?" Notes may have been retracted, but the value of the notes can't be guaranteed.

Each numbered key may appear multiple times with a different number suffix. For example, DISPENSED1, DISPENSED2, DISPENSED3. The number will start at 1 and increment. Each key can only be given once. Each key must have a value in a different currency. For example, DISPENSED1=100.00EUR,DISPENSED2=200.00USD

Note: In most cases, once currency has been presented and is accessible to a customer then the value of that currency shouldn't be relied on since the customer might take notes, replace notes with counterfeit etc. The value of any notes retracted back into the machine isn't reliable so RETRACTEDAMOUNT1 will be "?".

However, some hardware may be able to test notes that are retracted are valid. This is

possible with cash accepting hardware. The RETRACTEDAMOUNT value will only give an actual value if the notes have been checked by the hardware.

Note: Values in the PresentStatus token are the actual values. This is different to the token in the Dispense command where currency "up to" the token value may be dispensed. This doesn't apply to the PresentStatus token.

Event Messages

None

5.2.4 - Dispenser.Denominate

This command provides a denomination. A denomination specifies the number of items which are required from each cash unit in order to satisfy a given amount. The denomination depends upon the currencies, the mix algorithm and any partial denomination supplied by the client.

This command can also be used to validate that any denomination supplied by the client can be dispensed.

If items of differing currencies are to be included in the same denomination then the currencies array has one entry per currency. Alternatively the currency information can be omitted and the mix number must be 0 ("individual").

If the *cashBox* field returned by the Dispenser.Capabilites command is TRUE then, if the entire denomination cannot be satisfied, a partial denomination will be returned with the remaining amount to be supplied from the teller's cash box.

This command can be used in four different ways:

1. In order to check that it is possible to dispense a given denomination. The input parameters to the command are currency and denomination, with a mix number of 0 ("individual") and an amount of zero. If items of differing currencies are to be dispensed then the currencies array needs one item per currency.
2. In order to validate that a given amount matches a given denomination and that it is possible to dispense the denomination. The input parameters to the command should be amount, currency and denomination, with a mix number of 0 ("individual").
3. In order to obtain a denomination of a given amount. The input parameters supplied should be amount, currency and mix number.
4. In order to complete a partial denomination of a given amount. In this case the input parameters to the command should be currency, amount, mix number and either a partially specified denomination or a minimum amount from the cash

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

box. A completed denomination is returned. *cashBox* of the denomination structure may be updated as a result of this command.

Command Message

| Payload | Type | Required |
|---------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "tellerID": 0, | integer | |
| "mixNumber": 0, | integer | |
| "denomination": { | object | |
| "currencies": { | object | |
| "additionalProperties": 0 | number | |
| }, | | |
| "values": { | object | |
| "additionalProperties": 0 | integer | |
| }, | | |
| "cashBox": 0 | integer | |
| } | | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

tellerID

Identification of teller. This field is ignored if the device is a Self-Service Dispenser.

mixNumber

Mix algorithm or house mix table to be used.

default: 0

denomination

Denomination object describing the contents of the denomination operation.

denomination/currencies

"List of currency and amount combinations for denomination. There will be one entry for each currency in the denomination. The property name is the currency name in ISO format (e.g. "EUR").

denomination/currencies/additionalProperties

The amount to be denominated or dispensed. Use currency identifier in ISO format as property name.

denomination/values

This list specifies the number of items to take from the cash units. Each entry uses a cashunit object name as stated by the [CashManagement.GetCashUnitInfo](#) command. The value of the entry is the number of items to take from that unit. If the client does not wish to specify a denomination, it should omit the values property.

denomination/values/additionalProperties

Number of items to take from the specified cash unit. Use cash unit name as specified by the [CashManagement.GetCashUnitInfo](#) command as property name.

denomination/cashBox

Only applies to Teller Dispensers. Amount to be paid from the teller's cash box.

Completion Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "invalidCurrency", | string | |
| "currencies": { | object | |
| "additionalProperties": 0 | number | |
| }, | | |
| "values": { | object | |
| "additionalProperties": 0 | integer | |
| }, | | |
| "cashBox": 0 | integer | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

- invalidCurrency - There are no cash units in the device of the currency specified in one of the *currencyID* fields of the input structure.
- invalidTellerID - Invalid teller ID. This error will never be generated by a Self-Service device.

- cashUnitError - There is a problem with a cash unit. A `CashManagement.CashUnitErrorEvent` will be posted with the details.
- invalidDenomination - The `mixNumer` is individual and the sum of the values for `cashBox` and the items specified by `values` does not match the non-zero amount specified. This error code is not used when the amount specified is zero.
- invalidMixNumber - Unknown mix algorithm.
- noCurrencyMix - The cash units specified in the denomination were not all of the same currency and this device does not support multiple currencies.
- notDispensable - The amount is not dispensable by the device. This error code is also returned if the `mixNumber` is specified as individual, but a cash unit is specified in the `values` list which is not a dispensing cash unit, e.g., a retract/reject cash unit.
- tooManyItems - The request requires too many items to be dispensed.
- exchangeActive - The device is in an exchange state (see `CashManagement.StartExchange`).
- noCashBoxPresent - Cash box amount needed, however teller is not assigned a cash box.
- amountNotInMixTable - A mix table is being used to determine the denomination but the amount specified for the denomination is not in the mix table.

currencies

"List of currency and amount combinations for denomination. There will be one entry for each currency in the denomination. The property name is the currency name in ISO format (e.g. "EUR").

currencies/additionalProperties

The amount to be denominated or dispensed. Use currency identifier in ISO format as property name.

values

This list specifies the number of items to take from the cash units. Each entry uses a `cashunit` object name as stated by the [CashManagement.GetCashUnitInfo](#) command. The value of the entry is the number of items to take from that unit. If the client does not wish to specify a denomination, it should omit the `values` property.

values/additionalProperties

Number of items to take from the specified cash unit. Use cash unit name as specified by the [CashManagement.GetCashUnitInfo](#) command as property name.

cashBox

Only applies to Teller Dispensers. Amount to be paid from the teller's cash box.

Event Messages

- [CashManagement.CashUnitErrorEvent](#)

[5.2.5 - Dispenser.Dispense](#)

This command performs the dispensing of items to the customer. The command provides the same functionality as the Dispenser.Denominate command plus the additional functionality of dispensing the items. If items of differing currencies are to be dispensed then the currencies array has one entry per currency. Alternatively the currency information can be omitted and the mix number must be 0 ("individual"). However, these restrictions do not apply if a single currency is dispensed with non-currency items, such as coupons.

The Dispenser.Dispense command can be used in the following ways:

1. The input parameters to the command are amounts, currencies and denomination. The mix number is 0 ("individual"). In this case, the denomination is checked for validity and, if valid, is dispensed.
2. The input parameters are amounts, currencies and mix number. In this case the amount is denominated and, if this succeeds, the items are dispensed.
3. If the amount and currency information is omitted and a denomination is supplied with a mix number of 0 ("individual") the denomination is checked for validity and, if valid, is dispensed.
4. The command will calculate a partial denomination of a given amount and dispense the complete denomination. In this case the input parameters to the command should be currencies, amounts, mix number and either a partially specified denomination or a minimum amount from the cash box. The cash box amount may be updated as a result of this command.

If the *cashBox* field returned by the Dispenser.Capabilities command is TRUE then, if the entire denomination cannot be satisfied, a partial denomination will be returned with the remaining amount to be supplied from the teller's cash box.

If the device is a Teller Dispenser, the input field *position* can be set to "default". If this is the case the *tellerID* is used to perform the dispense operation to the assigned teller position.

It will be necessary to use the Dispenser.Present command to present the items to the user. If the Dispenser does not have an intermediate stacker the Dispenser.Present command does not need to be called and does not serve any purpose.

Note that a level 4 note can be dispensed, but is not necessarily presented to the customer. e.g. a note can be skewed, or can be unfit for dispensing.

The values in the completion message report the amount dispensed and the number of items dispensed from each cash unit.

Command Message

| Payload | Type | Required |
|------------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "tellerID": 0, | integer | |
| "mixNumber": 0, | integer | |
| "position": "default", | string | |
| "denomination": { | object | |
| "currencies": { | object | |
| "additionalProperties": 0 | number | |
| }, | | |
| "values": { | object | |
| "additionalProperties": 0 | integer | |
| }, | | |
| "cashBox": 0 | integer | |
| }, | | |
| "token": Add example to YAML | string | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

tellerID

Identifies the teller. This field is ignored if the device is a Self-Service Dispenser.

mixNumber

Mix algorithm or house mix table to be used to create a denomination of the supplied amount. If the value is 0 ("individual"), the denomination supplied in the *denomination* field is validated prior to the dispense operation. If it is found to be invalid no alternative denomination will be calculated.

default: 0

position

Required output position. Following values are possible:

- default - The default configuration information is used. This can be either position dependent or teller dependent.
- left - Present items to left side of device.
- right - Present items to right side of device.
- center - Present items to center output position.
- top - Present items to the top output position.
- bottom - Present items to the bottom output position.
- front - Present items to the front output position.
- rear - Present items to the rear output position.

denomination

Denomination object describing the denominations used for the dispense operation.

denomination/currencies

"List of currency and amount combinations for denomination. There will be one entry for each currency in the denomination. The property name is the currency name in ISO format

(e.g. "EUR").

denomination/currencies/additionalProperties

The amount to be denominated or dispensed. Use currency identifier in ISO format as property name.

denomination/values

This list specifies the number of items to take from the cash units. Each entry uses a cashunit object name as stated by the [CashManagement.GetCashUnitInfo](#) command. The value of the entry is the number of items to take from that unit. If the client does not wish to specify a denomination, it should omit the values property.

denomination/values/additionalProperties

Number of items to take from the specified cash unit. Use cash unit name as specified by the [CashManagement.GetCashUnitInfo](#) command as property name.

denomination/cashBox

Only applies to Teller Dispensers. Amount to be paid from the teller's cash box.

token

The dispense token that authorises the dispense operation, as created by the authorising host. See the section on end to end security for more information.

The dispense token will follow the standard token format, and will contain the following key:

"DISPENSE1": The maximum value to be dispensed. This will be a number string that may contain a fractional part. The decimal character will be ". ". The value, including the fractional part, will be defined by the ISO currency. The number will be followed by the ISO currency code. The currency code will be upper case.

For example, "123.45EUR" will be €123 and 45 cents.

The "DISPENSE" key may appear multiple times with a number suffix. For example, DISPENSE1, DISPENSE2, DISPENSE3. The number will start at 1 and increment. Each key can only be given once. Each key must have a value in a different currency. For example, DISPENSE1=100.00EUR,DISPENSE2=200.00USD

The actual amount dispensed will be given by the denomination. The value in the token

MUST be greater or equal to the amount in the denomination parameter. If the Token has a lower value, or the Token is invalid for any reason, then the command will fail with an invalid data error code.

Completion Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "invalidCurrency", | string | |
| "currencies": { | object | |
| "additionalProperties": 0 | number | |
| }, | | |
| "values": { | object | |
| "additionalProperties": 0 | integer | |
| }, | | |
| "cashBox": 0 | integer | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

- invalidCurrency - There are no cash units in the device of the currency specified in one of the *currencyID* fields of the input structure.
- invalidTellerID - Invalid teller ID. This error will never be generated by a Self-Service device.
- cashUnitError - There is a problem with a cash unit. A *CashManagement.CashUnitErrorEvent* will be posted with the details.
- invalidDenomination - The sum of the values for cash box and cash units was greater than the amount specified.
- invalidMixNumber - Unknown mix algorithm.
- noCurrencyMix - The cash units specified in the denomination were not all of the same currency and this device does not support multiple currencies.
- notDispensable - The amount is not dispensable by the device. This error code is also returned if the *mixNumber* is specified as individual, but a cash unit is specified in the *values* list which is not a dispensing cash unit, e.g., a retract/reject cash unit.
- tooManyItems - The request requires too many items to be dispensed. This error is also generated if sub-dispensing is required.
- unsupportedPosition - The specified output position is not supported.
- exchangeActive - The device is in an exchange state (see *CashManagement.StartExchange*)
- noCashBoxPresent - Cash box amount needed, however teller is not assigned a cash box.
- amountNotInMixTable - A mix table is being used to determine the denomination but the amount specified for the denomination is not in the mix table.
- itemsNotTaken - Items have not been taken during a sub-dispense operation. This error occurs if a hardware timeout expires.
- itemsLeft - Items have been left in the transport or exit slot as a result of a prior dispense, present or recycler cash-in operation.
- shutterOpen - The Service cannot dispense items with an open output shutter.

currencies

"List of currency and amount combinations for denomination. There will be one entry for each currency in the denomination. The property name is the currency name in ISO format (e.g. "EUR").

currencies/additionalProperties

The amount to be denominated or dispensed. Use currency identifier in ISO format as property name.

values

This list specifies the number of items to take from the cash units. Each entry uses a cashunit object name as stated by the [CashManagement.GetCashUnitInfo](#) command. The value of the entry is the number of items to take from that unit. If the client does not wish to specify a denomination, it should omit the values property.

values/additionalProperties

Number of items to take from the specified cash unit. Use cash unit name as specified by the [CashManagement.GetCashUnitInfo](#) command as property name.

cashBox

Only applies to Teller Dispensers. Amount to be paid from the teller's cash box.

Event Messages

- [CashManagement.CashUnitThresholdEvent](#)
- [Dispenser.DelayedDispenseEvent](#)
- [Dispenser.StartDispenseEvent](#)
- [CashManagement.CashUnitErrorEvent](#)
- [Dispenser.ItemsTakenEvent](#)
- [Dispenser.PartialDispenseEvent](#)
- [Dispenser.SubDispenseOkEvent](#)
- [Dispenser.IncompleteDispenseEvent](#)
- [CashManagement.NoteErrorEvent](#)
- [CashManagement.InfoAvailableEvent](#)
- [Dispenser.ShutterStatusChangedEvent](#)

[5.2.6 - Dispenser.Present](#)

This command will move items to the exit position for removal by the user. If a shutter exists, then it will be implicitly controlled during the present operation, even if the *shutterControl* capability is set to FALSE. The shutter will be closed when the user removes the items or the items are retracted. If *position* is "default" the position set in the Dispenser.Dispense command which caused these items to be dispensed will be used.

When this command successfully completes the items are in customer access.

If Dispenser.Present was called as part of a sub-dispense operation, the completion message includes details about remaining bunches. The field *additionalBunches* specifies, if

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

there are any additional bunches to be dispensed to the customer and the field *bunchesRemaining* specifies the number of outstanding subdispense operations.

Command Message

| Payload | Type | Required |
|-----------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "position": "default" | string | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

position

Output position where the amount is to be presented. Following values are possible:

- default - The default configuration.
- left - The left output position.
- right - The right output position.
- center - The center output position.
- top - The top output position.
- bottom - The bottom output position.
- front - The front output position.
- rear - The rear output position.

Completion Message

| Payload | Type | Required |
|------------------------------|--------|----------|
| { | | |
| "completionCode": "success", | string | |

```
"errorDescription": Add example to YAML, string  
"errorCode": "shutterNotOpen", string  
"position": "default", string  
"additionalBunches": "none", string  
"bunchesRemaining": 0 integer  
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

- shutterNotOpen - The shutter did not open when it should have. No items presented.
- shutterOpen - The shutter is open when it should be closed. No items presented.
- noItems - There are no items on the stacker.
- exchangeActive - The device is in an exchange state (see CashManagement.StartExchange).
- presentErrorNoItems - There was an error during the present operation - no items were presented.
- presentErrorItems - There was an error during the present operation - at least some of the items were presented.
- presentErrorUnknown - There was an error during the present operation - the position of the items is unknown. Intervention may be required to reconcile the cash amount totals.
- unsupportedPosition - The position specified is not supported.

position

Specifies the position where the items have been presented. Following values are possible:

- left - Items presented at the left output position.
- right - Items presented at the right output position.
- center - Items presented at the center output position.
- top - Items presented at the top output position.
- bottom - Items presented at the bottom output position.
- front - Items presented at the front output position.
- rear - Items presented at the rear output position.

additionalBunches

Specifies whether or not additional bunches of items are remaining to be presented as a result of the current operation. Following values are possible:

- none - No additional bunches remain.
- oneMore - At least one additional bunch remains.
- unknown - It is unknown whether additional bunches remain.

bunchesRemaining

If *additionalBunches* is "oneMore", specifies the number of additional bunches of items remaining to be presented as a result of the current operation. If the number of additional bunches is at least one, but the precise number is unknown, *bunchesRemaining* will be 255 (TODO: Check if there is a better way to represent this state). For any other value of *additionalBunches*, *bunchesRemaining* will be zero.

Event Messages

- [CashManagement.CashUnitThresholdEvent](#)
- [Dispenser.ItemsTakenEvent](#)
- [CashManagement.InfoAvailableEvent](#)
- [Dispenser.ShutterStatusChangedEvent](#)

[5.2.7 - Dispenser.Reject](#)

This command will move items from the intermediate stacker and transport them to a reject cash unit (i.e. a cash unit with type "rejectCassette"). The *count* field of the reject cash unit is incremented by the number of items that were thought to be present at the time of the reject or the number counted by the device during the reject. Note that the reject bin count is unreliable.

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Command Message

| Payload | Type | Required |
|-----------------|---------|----------|
| { | | |
| "timeout": 5000 | integer | |
| } | | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "cashUnitError" | string | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

- **cashUnitError** - A reject cash unit caused a problem. A `CashManagement.CashUnitErrorEvent` will be posted with the details.
- **noItems** - There were no items on the stacker.
- **exchangeActive** - The device is in an exchange state (see `CashManagement.StartExchange`).

Event Messages

- `CashManagement.CashUnitThresholdEvent`
- `CashManagement.CashUnitErrorEvent`
- `CashManagement.InfoAvailableEvent`

5.2.8 - Dispenser.Retract

This command will retract items which may have been in customer access from an output position or from internal areas within the Dispenser. Retracted items will be moved to either a retract cash unit, a reject cash unit, item cash units, the transport or the intermediate stacker. After the items are retracted the shutter is closed automatically, even if the *shutterControl* capability is set to FALSE.

If items are moved to a retract cash unit (i.e. a cash unit with *type* "retractCassette"), then the *count* field of the retract cash unit must be incremented by 1 to specify the number of retracts. If items are moved to any other cash unit (e.g. a cash unit with *type* "rejectCassette") then the *count* field of the cash unit must be incremented by the number of items that were thought to be present at the time the `Dispenser.Retract` command was issued or the number counted by the device during the retract. Note that reject bin counts are unreliable.

Command Message

| Payload | Type | Required |
|------------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "outputPosition": "default", | string | |
| "retractArea": "retract", | string | |

```
"index": 0          integer  
}  
  
Properties
```

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

outputPosition

Output position from which to retract the items. Following values are possible:

- default - The default configuration information should be used.
- left - Retract items from the left output position.
- right - Retract items from the right output position.
- center - Retract items from the center output position.
- top - Retract items from the top output position.
- bottom - Retract items from the bottom output position.
- front - Retract items from the front output position.
- rear - Retract items from the rear output position.

retractArea

This value specifies the area to which the items are to be retracted. Following values are possible:

- retract - Retract the items to a retract cash unit.
- transport - Retract the items to the transport.
- stacker - Retract the items to the intermediate stacker area.
- reject - Retract the items to a reject cash unit.
- itemCassette - Retract the items to the item cassettes, i.e. cassettes that can be dispensed from.

index

If *retractArea* is set to "retract" this field defines the position inside the retract cash units into which the cash is to be retracted. *index* starts with a value of one (1) for the first retract position and increments by one for each subsequent position. If there are several retract

cash units (of type "retractCassette" in command CashManagement.CashUnitInfo), *index* would be incremented from the first position of the first retract cash unit to the last position of the last retract cash unit. The maximum value of *index* is the sum of *maximum* of each retract cash unit. If *retractArea* is not set to "retract" the value of this field is ignored.

Completion Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "cashUnitError", | string | |
| "itemNumber": [{ | array (object) | |
| "currencyID": Add example to YAML, | string | |
| "values": 0, | number | |
| "release": 0, | integer | |
| "count": 0, | integer | |
| "cashunit": Add example to YAML | string | |
| }] | | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

- cashUnitError - There is a problem with a cash unit. A `CashManagement.CashUnitErrorEvent` will be posted with the details.
- noItems - There were no items to retract.
- exchangeActive - The device is in an exchange state (see `CashManagement.StartExchange`).
- shutterNotClosed - The shutter failed to close.
- itemsTaken - Items were present at the output position at the start of the operation, but were removed before the operation was complete - some or all of the items were not retracted.
- invalidRetractPosition - The *index* is not supported.
- notRetractArea - The retract area specified in *retractArea* is not supported.
- unsupportedPosition - The retract area specified in *retractArea* is not empty so the retract operation is not possible.
- positionNotEmpty - The request requires too many items to be dispensed.
- incompleteRetract - Some or all of the items were not retracted for a reason not covered by other error codes. The detail will be reported with the `Dispenser.IncompleteRetractEvent`.

itemNumber

Array of item number objects.

itemNumber/currencyID

A three character array storing the ISO format [Ref. 2] Currency ID; if the currency of the item is not known this is omitted.

itemNumber/values

The value of a single item expressed as floating point value; or a zero value if the value of the item is not known.

itemNumber/release

The release of the item. The higher this number is, the newer the release. Zero means that there is only one release or the release is not known. This value has not been standardized and therefore a release number of the same item will not necessarily have the same value in different systems.

itemNumber/count

The count of items of the same type moved to the same destination during the execution of this command.

itemNumber/cashunit

The object name of the cash unit which received items during the execution of this command as stated by the [CashManagement.GetCashUnitInfo](#) command. This value will be omitted if items were moved to the [retractArea](#) transport or stacker.

Event Messages

- [CashManagement.CashUnitThresholdEvent](#)
- [CashManagement.CashUnitErrorEvent](#)
- [Dispenser.ItemsTakenEvent](#)
- [CashManagement.InfoAvailableEvent](#)
- [Dispenser.IncompleteRetractEvent](#)
- [Dispenser.ShutterStatusChangedEvent](#)

[5.2.9 - Dispenser.OpenShutter](#)

This command opens the shutter.

Command Message

| Payload | Type | Required |
|-----------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "position": "default" | string | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

timeout but can be cancelled.

default: 0

position

The output position where the shutter is to be opened. If the client does not need to specify a shutter, this field can be omitted or its contents set to "default". Following values are possible:

- default - The default configuration information should be used.
- left - Open the shutter at the left output position.
- right - Open the shutter at the right output position.
- center - Open the shutter at the center output position.
- top - Open the shutter at the top output position.
- bottom - Open the shutter at the bottom output position.
- front - Open the shutter at the front output position.
- rear - Open the shutter at the rear output position.

Completion Message

| Payload | Type | Required |
|--|----------------------------|----------|
| { "completionCode": "success", "errorDescription": Add example to YAML, "errorCode": "unsupportedPosition" } | string string string | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

- unsupportedPosition - The position specified is not supported.
- shutterNotOpen - The shutter failed to open.
- shutterOpen - The shutter was already open.
- exchangeActive - The device is in an exchange state (see CashManagement.StartExchange).

Event Messages

- Dispenser.ShutterStatusChangedEvent

5.2.10 - Dispenser.CloseShutter

This command closes the shutter.

Command Message

| Payload | Type | Required |
|---|-------------------|----------|
| { "timeout": 5000, "position": "default" } | integer string | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

position

The output position where the shutter is to be closed. If the client does not need to specify a

shutter, this field can be omitted or its contents set to "default". Following values are possible:

- default - The default configuration information should be used.
- left - Close the shutter at the left output position.
- right - Close the shutter at the right output position.
- center - Close the shutter at the center output position.
- top - Close the shutter at the top output position.
- bottom - Close the shutter at the bottom output position.
- front - Close the shutter at the front output position.
- rear - Close the shutter at the rear output position.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "unsupportedPosition" | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

- unsupportedPosition - The position specified is not supported.
- shutterClosed - The shutter was already closed.

- `shutterNotClosed` - The shutter failed to close.
- `exchangeActive` - The device is in an exchange state (see [CashManagement.StartExchange](#)).

Event Messages

- [Dispenser.ShutterStatusChangedEvent](#)

5.2.11 - Dispenser.SetMixTable

This command is used to set up the mix table specified by the mix number. Mix tables are persistent and are available to all clients in the system. An amount can be specified as different denominations within the mix table. If the amount is specified more than once the Service Provider will attempt to denominate or dispense the first amount in the table. If this does not succeed (e.g. because of a cash unit failure) the Service Provider will attempt to denominate or dispense the next amount in the table. The Service Provider can only dispense amounts which are explicitly mentioned in the mix table.

If a mix number passed in already exists then the information is overwritten with the new information.

Command Message

| Payload | Type | Required |
|------------------------------|-----------------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "mixNumber": 0, | integer | |
| "name": Add example to YAML, | string | |
| "mixHeader": [0], | array (number) | |
| "mixRows": [{ | array (object) | |
| "amount": 0, | number | |
| "mixture": [0] | array (integer) | |
| }] | | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

mixNumber

Number identifying the house mix table.

name

Name of the house mix table.

mixHeader

Array of floating point numbers; each element defines the value of the item corresponding to its respective column.

mixRows

Array of rows of the mix table.

mixRows/amount

Amount denominated by this mix row.

mixRows/mixture

A mix row, an array of integers; each element defines the quantity of each item denomination in the mix used in the denomination of *amount*. The value of each array element is defined by the *mixHeader*.

Completion Message

Payload

Type Required

```
{  
  "completionCode": "success",           string  
  "errorDescription": Add example to YAML, string  
  "errorCode": "invalidMixNumber"        string  
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

- invalidMixNumber - The *mixNumber* is reserved for a predefined mix algorithm.
- invalidMixTable - The contents of at least one of the defined rows of the mix table is incorrect.

Event Messages

None

5.2.12 - Dispenser.Reset

This command is used by the client to perform a hardware reset which will attempt to return the Dispenser device to a known good state. This command does not over-ride a lock obtained through Common.Lock (TODO) on another client or service handle.

The device will attempt to move any items found anywhere within the device to the position specified within the command payload. This may not always be possible because of hardware problems.

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

If items are found inside the device the Dispenser.MediaDetectedEvent will be generated and will inform the client where the items were actually moved to.

If an exchange state is active then this command will end the exchange state (even if this command does not complete successfully).

On a recycling device this command is not accepted if a cash-in transaction is active and will return a "deviceNotReady" error.

If items are moved to a retract cash unit (i.e. a cash unit with *type* "retractCassette"), then the *count* field of the retract cash unit must be incremented by 1 to specify the number of operations that changed the count. If items are moved to any other cash unit (e.g. a cash unit with *type* "rejectCassette"), then the *count* field of the cash unit must be incremented either by the number of items that were present at the time the Dispenser.Reset command was issued or the number counted by the device during the Dispenser.Reset command. Note that reject bin counts are unreliable.

Command Message

| Payload | Type | Required |
|----------------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "cashunit": Add example to YAML, | string | |
| "retractArea": { | object | |
| "outputPosition": "default", | string | |
| "retractArea": "retract", | string | |
| "index": 0 | integer | |
| }, | | |
| "outputPosition": "default" | string | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

cashunit

If defined, this value specifies the object name (as stated by the [CashManagement.GetCashUnitInfo](#) command) of the single cash unit to be used for the storage of any items found.

retractArea

This field is used if items are to be moved to internal areas of the device, including cash units, the intermediate stacker, or the transport.

retractArea/outputPosition

Output position from which to retract the items. Following values are possible:

- default - The default configuration information should be used.
- left - Retract items from the left output position.
- right - Retract items from the right output position.
- center - Retract items from the center output position.
- top - Retract items from the top output position.
- bottom - Retract items from the bottom output position.
- front - Retract items from the front output position.
- rear - Retract items from the rear output position.

retractArea/retractArea

This value specifies the area to which the items are to be retracted. Following values are possible:

- retract - Retract the items to a retract cash unit.
- transport - Retract the items to the transport.
- stacker - Retract the items to the intermediate stacker area.
- reject - Retract the items to a reject cash unit.
- itemCassette - Retract the items to the item cassettes, i.e. cassettes that can be dispensed from.

retractArea/index

If *retractArea* is set to "retract" this field defines the position inside the retract cash units into which the cash is to be retracted. *index* starts with a value of one (1) for the first retract position and increments by one for each subsequent position. If there are several retract

cash units (of type "retractCassette" in command CashManagement.CashUnitInfo), *index* would be incremented from the first position of the first retract cash unit to the last position of the last retract cash unit. The maximum value of *index* is the sum of *maximum* of each retract cash unit. If *retractArea* is not set to "retract" the value of this field is ignored.

outputPosition

The output position to which items are to be moved. This field is only used if *number* is zero and *retractArea* is omitted. Following values are possible:

- default - The default configuration.
- left - The left output position.
- right - The right output position.
- center - The center output position.
- top - The top output position.
- bottom - The bottom output position.
- front - The front output position.
- rear - The rear output position.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "cashUnitError" | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

- cashUnitError - There is a problem with a cash unit. A `CashManagement.CashUnitErrorEvent` will be posted with the details.
- unsupportedPosition - The position specified is not supported.
- invalidCashUnit - The cash unit number specified is not valid.
- invalidRetractPosition - The *index* is not supported.
- notRetractArea - The retract area specified in *retractArea* is not supported.
- positionNotEmpty - The retract area specified in *retractArea* is not empty so the moving of items was not possible.
- incompleteRetract - Some or all of the items were not retracted for a reason not covered by other error codes. The detail will be reported with the `Dispenser.IncompleteRetractEvent`.

Event Messages

- `CashManagement.CashUnitThresholdEvent`
- `CashManagement.CashUnitErrorEvent`
- `Dispenser.MediaDetectedEvent`
- `Dispenser.ItemsTakenEvent`
- `CashManagement.InfoAvailableEvent`
- `Dispenser.IncompleteRetractEvent`
- `Dispenser.ShutterStatusChangedEvent`

5.2.13 - Dispenser.TestCashUnits

This command is used to test cash units following replenishment. The command payload specifies where items dispensed as a result of this command should be moved to. All cash units which are testable (i.e. that have a *status* of "ok" or "low" and no client lock in the cash unit) are tested. If the hardware is able to do so tests are continued even if an error occurs while testing one of the cash units. The command completes with success completion message if the Service successfully manages to test all of the testable cash units regardless of the outcome of the test. This is the case if all testable cash units could be tested and a dispense was possible from at least one of the cash units.

A `CashManagement.CashUnitErrorEvent` will be sent for any cash unit which cannot be tested or which failed the test. **If all the cash units could not be tested or no cash units are testable then a "cashUnitError" code will be returned and** `CashManagement.CashUnitErrorEvents` generated for every cash unit that encountered a

problem. The operation performed to test the cash units is vendor dependent. Items may be dispensed or transported into a reject bin as a result of this command.

If no cash units are testable then a "cashUnitError" code will be returned and `CashManagement.CashUnitErrorEvents` will be generated for every cash unit.

Command Message

| Payload | Type | Required |
|---|---------|----------|
| { | | |
| " timeout<td>integer</td><td></td> | integer | |
| " cashunit<td>string</td><td></td> | string | |
| " retractArea<td>object</td><td></td> | object | |
| " outputPosition<td>string</td><td></td> | string | |
| " retractArea<td>string</td><td></td> | string | |
| " index<td>integer</td><td></td> | integer | |
| }, | | |
| " outputPosition<td>string</td><td></td> | string | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

cashunit

If defined, this value specifies the object name (as stated by the `CashManagement.GetCashUnitInfo` command) of the single cash unit to be used for the storage of any items found.

retractArea

This field is used if items are to be moved to internal areas of the device, including cash units, the intermediate stacker, or the transport.

retractArea/outputPosition

Output position from which to retract the items. Following values are possible:

- default - The default configuration information should be used.
- left - Retract items from the left output position.
- right - Retract items from the right output position.
- center - Retract items from the center output position.
- top - Retract items from the top output position.
- bottom - Retract items from the bottom output position.
- front - Retract items from the front output position.
- rear - Retract items from the rear output position.

retractArea/retractArea

This value specifies the area to which the items are to be retracted. Following values are possible:

- retract - Retract the items to a retract cash unit.
- transport - Retract the items to the transport.
- stacker - Retract the items to the intermediate stacker area.
- reject - Retract the items to a reject cash unit.
- itemCassette - Retract the items to the item cassettes, i.e. cassettes that can be dispensed from.

retractArea/index

If *retractArea* is set to "retract" this field defines the position inside the retract cash units into which the cash is to be retracted. *index* starts with a value of one (1) for the first retract position and increments by one for each subsequent position. If there are several retract cash units (of type "retractCassette" in command CashManagement.CashUnitInfo), *index* would be incremented from the first position of the first retract cash unit to the last position of the last retract cash unit. The maximum value of *index* is the sum of *maximum* of each retract cash unit. If *retractArea* is not set to "retract" the value of this field is ignored.

outputPosition

The output position to which items are to be moved. This field is only used if *number* is zero

and retractArea is omitted. Following values are possible:

- default - The default configuration.
- left - The left output position.
- right - The right output position.
- center - The center output position.
- top - The top output position.
- bottom - The bottom output position.
- front - The front output position.
- rear - The rear output position.

Completion Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "cashUnitError", | string | |
| "cashunits": { | object | |
| "additionalProperties": { | object | |
| "status": "ok", | string | |
| "type": "billCassette", | string | |
| "currencyID": Add example to YAML, | string | |
| "value": 0, | number | |
| "logicalCount": 0, | integer | |
| "maximum": 0, | integer | |
| "appLock": false, | boolean | |
| "cashUnitName": Add example to YAML, | string | |
| "initialCount": 0, | integer | |
| "dispensedCount": 0, | integer | |
| "presentedCount": 0, | integer | |

| | |
|--|-----------------|
| "retractedCount": 0, | integer |
| "rejectCount": 0, | integer |
| "minimum": 0, | integer |
| "physicalPositionName": Add example to YAML, | string |
| "unitID": Add example to YAML, | string |
| "count": 0, | integer |
| "maximumCapacity": 0, | integer |
| "hardwareSensor": false, | boolean |
| "itemType": { | object |
| "all": false, | boolean |
| "unfit": false, | boolean |
| "individual": false, | boolean |
| "level1": false, | boolean |
| "level2": false, | boolean |
| "level3": false, | boolean |
| "itemProcessor": false, | boolean |
| "unfitIndividual": false | boolean |
| } | |
| "cashInCount": 0, | integer |
| "noteNumberList": { | object |
| "noteNumber": [{ | array (object) |
| "noteID": 0, | integer |
| "count": 0 | integer |
| }] | |
| } | |
| "noteIDs": [0] | array (integer) |

```
}
```

```
}
```

```
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

- cashUnitError - A cash unit caused a problem that meant all cash units could not be tested or no cash units were testable. One or more CashManagement.CashUnitErrorEvent will be posted with the details.
- unsupportedPosition - The position specified is not supported.
- shutterNotOpen - The shutter is not open or did not open when it should have. No items presented.
- shutterOpen - The shutter is open when it should be closed. No items presented.
- invalidCashUnit - The cash unit number specified is not valid.
- exchangeActive - The device is in an exchange state (see CashManagement.StartExchange).
- presentErrorNoItems - There was an error during the present operation - no items were presented.
- presentErrorItems - There was an error during the present operation - at least some of the items were presented.
- presentErrorUnknown - There was an error during the present operation - the position of the items is unknown. Intervention may be required to reconcile the cash amount totals.

cashunits

Object containing cash unit information.

cashunits/additionalProperties

Cash unit information.

cashunits/additionalProperties/status

Supplies the status of the cash unit. Following values are possible:

- ok - The cash unit is in a good state.
- full - The cash unit is full.
- high - The cash unit is almost full (i.e. reached or exceeded the threshold defined by *maximum*).
- low - The cash unit is almost empty (i.e. reached or below the threshold defined by *minimum*).
- empty - The cash unit is empty, or insufficient items in the cash unit are preventing further dispense operations.
- inoperative - The cash unit is inoperative.
- missing - The cash unit is missing.
- noValue - The values of the specified cash unit are not available.
- noReference - There is no reference value available for the notes in this cash unit. The cash unit has not been calibrated.
- manipulated - The cash unit has been inserted (including removal followed by a reinsertion) when the device was not in the exchange state. This cash unit cannot be dispensed from.

cashunits/additionalProperties/type

Type of cash unit. Following values are possible:

- notApplicable - Not applicable. Typically means cash unit is missing.
- rejectCassette - Reject cash unit. This type will also indicate a combined reject/retract cash unit.
- billCassette - Cash unit containing bills.
- coinCylinder - Coin cylinder.
- coinDispenser - Coin dispenser as a whole unit.
- retractCassette - Retract cash unit.
- coupon - Cash unit containing coupons or advertising material.
- document - Cash unit containing documents.
- replenishmentContainer - Replenishment container. A cash unit can be refilled from a replenishment container.
- recycling - Recycling cash unit. This unit is only present when the device implements the Dispenser and CashAcceptor interfaces.

- cashIn - Cash-in cash unit.

cashunits/additionalProperties/currencyID

A three character string storing the ISO format [Ref. 2] Currency ID. This value will be omitted for cash units which contain items of more than one currency type or items to which currency is not applicable. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

cashunits/additionalProperties/value

Supplies the value of a single item in the cash unit. This value is expressed as floating point value. If the *currencyID* field for this cash unit is omitted, then this field will contain zero. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

cashunits/additionalProperties/logicalCount

The meaning of this count depends on the type of cash unit. This value is persistent. For all cash units except retract cash units (*type* is not *retractCassette*) this value specifies the number of items inside the cash unit. For all dispensing cash units (*type* is *billCassette*, *coinCylinder*, *coinDispenser*, *coupon*, *document* or *recycling*), this value includes any items from the cash unit not yet presented to the customer. This count is only decremented when the items are either known to be in customer access or successfully rejected. If the cash unit is usable from the CashAcceptor interface (*type* is *recycling*, *cashIn*, *retractCassette* or *rejectCassette*) then this value will be incremented as a result of a cash-in operation. Note that for a reject cash unit (*type* is *rejectCassette*), this value is unreliable, since the typical reason for dumping items to the reject cash unit is a suspected count failure. For a retract cash unit (*type* is *retractCassette*) this value specifies the number of retract operations which result in items entering the cash unit.

cashunits/additionalProperties/maximum

When *count* reaches this value the threshold event

CashManagement.CashUnitThresholdEvent (*high*) will be generated. This value can be different from the actual capacity of the cassette. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

cashunits/additionalProperties/appLock

If this value is TRUE items cannot be dispensed from or deposited into the cash unit. If this value is TRUE and the client attempts to use the cash unit a

CashManagement.CashUnitErrorEvent event will be generated and an error completion message will be returned. This value is persistent.

cashunits/additionalProperties/cashUnitName

A name which helps to identify the type of the cash unit. This is especially useful in the case of cash units of type *document* where different documents can have the same currency and value. For example, travelers checks and bank checks may have the same currency and value but still need to be identifiable as different types of document. Where this value is not relevant (e.g. in bill cash units) the property can be omitted. This value is persistent.

cashunits/additionalProperties/initialCount

Initial number of items contained in the cash unit. This value is persistent.

cashunits/additionalProperties/dispensedCount

The number of items dispensed from this cash unit. This count is incremented when the items are removed from the cash units. This count includes any items that were rejected during the dispense operation and are no longer in this cash unit. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

cashunits/additionalProperties/presentedCount

The number of items from this cash unit that have been presented to the customer. This count is incremented when the items are presented to the customer. If it is unknown if a customer has been presented with the items, then this count is not updated. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

cashunits/additionalProperties/retractedCount

The number of items that have been accessible to a customer and retracted into the cash unit. This value is persistent.

cashunits/additionalProperties/rejectCount

The number of items dispensed from this cash unit which have been rejected, are in a cash unit other than this cash unit, and which have not been accessible to a customer. This value may be unreliable, since a typical reason for rejecting items is a suspected pick failure. Other reasons for rejecting items may include incorrect note denominations, classifications

not valid for dispensing, or where the transaction has been cancelled and a Reject command has been called. For reject and retract cash units (*type* is *rejectCassette* or *retractCassette*) this field does not apply and will be reported as zero. This value is persistent.

cashunits/additionalProperties/minimum

This field is not applicable to retract and reject cash units. For all cash units which dispense items (all other), when *count* reaches this value the threshold event CashManagement.CashUnitThresholdEvent (*low*) will be generated. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

cashunits/additionalProperties/physicalPositionName

A name identifying the physical location of the cash unit.

cashunits/additionalProperties/unitID

A 5 character string uniquely identifying the cash unit.

cashunits/additionalProperties/count

As defined by the *logicalCount* description, but with the following exceptions: This count does not include items dispensed but not yet presented. On cash units with *type* set to "retractCassette" the count represents the number of items, unless the device cannot count items during a retract, in which case this count will be zero. This value is persistent.

cashunits/additionalProperties/maximumCapacity

The maximum number of items the cash unit can hold. This is only for informational purposes. No threshold event CashManagement.CashUnitThresholdEvent will be generated. This value is persistent.

cashunits/additionalProperties/hardwareSensor

Specifies whether or not threshold events can be generated based on hardware sensors in the device. If this value is TRUE then threshold events may be generated based on hardware sensors as opposed to counts.

cashunits/additionalProperties/itemType

Specifies the type of items the cash unit takes as a combination of the following flags. The table in the Comments section of this command defines how to interpret the combination of these flags (TODO: include Table)

cashunits/additionalProperties/itemType/all

The cash unit takes all fit banknote types. These are level 4 notes which are fit for recycling.

cashunits/additionalProperties/itemType/unfit

The cash unit takes all unfit banknotes. These are level 4 notes which are unfit for recycling.

cashunits/additionalProperties/itemType/individual

The cash unit takes all types of fit banknotes specified in an individual list. These are level 4 notes which are fit for recycling.

cashunits/additionalProperties/itemType/level1

Level 1 note types are stored in this cash unit.

cashunits/additionalProperties/itemType/level2

If notes can be classified as level 2, then level 2 note types are stored in this cash unit.

cashunits/additionalProperties/itemType/level3

If notes can be classified as level 3, then level 3 note types are stored in this cash unit.

cashunits/additionalProperties/itemType/itemProcessor

The cash unit can accept items on the ItemProcessor interface.

cashunits/additionalProperties/itemType/unfitIndividual

The cash unit takes all types of unfit banknotes specified in an individual list. These are level 4 notes which are unfit for recycling.

cashunits/additionalProperties/cashInCount

Count of items that have entered the cash unit. This counter is incremented whenever an item enters a cash unit for any reason, unless it originated from this cash unit but was returned without being accessible to a customer. For a retract cash unit this value represents the total number of items of all types in the cash unit, or if the device cannot count items during a retract operation this value will be zero. This value is persistent.

cashunits/additionalProperties/noteNumberList

Array of cash items inside the cash unit. The content of this structure is persistent. If the cash unit is Dispenser specific cash unit with *type billCassette* or the contents of the cash unit are not known this structure will be omitted. If the cash unit is of *type retractCassette* this pointer will be omitted except for the following cases:

- If the retract cash unit is configured to accept level 2 notes then the number and type of level 2 notes is returned in the *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of level 2 notes.
- If items are recognized during retract operations then the number and type of notes retracted is returned in *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of retracted items.

cashunits/additionalProperties/noteNumberList/noteNumber

Array of banknote numbers the cash unit contains.

cashunits/additionalProperties/noteNumberList/noteNumber/noteID

Identification of note type. The Note ID represents the note identifiers reported by the *CashAcceptor.BanknoteTypes* command. If this value is zero then the note type is unknown.

cashunits/additionalProperties/noteNumberList/noteNumber/count

Actual count of cash items. The value is incremented each time cash items are moved to a cash unit. In the case of recycle cash units this count is decremented as defined in the description of the *logicalCount* field.

cashunits/additionalProperties/noteIDs

Array of integers which contains the note IDs of the banknotes the cash-in cash unit or recycle cash unit can take. This field only applies to *individual* cassette types. If there are no

note IDs defined for the cassette or the cassette is not defined as *individual* then *noteIDs* will be omitted.

Event Messages

- [CashManagement.CashUnitThresholdEvent](#)
- [CashManagement.CashUnitErrorEvent](#)
- [Dispenser.ItemsTakenEvent](#)
- [CashManagement.CashUnitInfoChangedEvent](#)
- [CashManagement.NoteErrorEvent](#)
- [Dispenser.ShutterStatusChangedEvent](#)
- [CashManagement.InfoAvailableEvent](#)

[5.2.14 - Dispenser.Count](#)

This command empties the specified cash unit(s). All items dispensed from the cash unit are counted and moved to the specified output location.

The number of items counted can be different from the number of items dispensed in cases where the Dispenser has the ability to detect this information. If the Dispenser cannot differentiate between what is dispensed and what is counted then *dispensed* will be the same as *counted*.

Upon successful Dispenser.Count command execution the cash unit(s) *count* field is reset.

Command Message

| Payload | Type | Required |
|---|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "emptyAll": false, | boolean | |
| "position": "default", | string | |
| "physicalPositionName": Add example to YAML | string | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

emptyAll

Specifies whether all cash units are to be emptied. If this value is TRUE then *physicalPositionName* is ignored.

position

Specifies the location to which items should be moved. Following values are possible:

- default - Output location is determined by Service.
- left - Present items to left side of device.
- right - Present items to right side of device.
- center - Present items to center output position.
- top - Present items to the top output position.
- bottom - Present items to the bottom output position.
- front - Present items to the front output position.
- rear - Present items to the rear output position.
- reject - Reject bin is used as output location.

physicalPositionName

Specifies which cash unit to empty and count. This name is the same as the *physicalPositionName* in the [CashManagement.GetCashUnitInfo](#) completion message.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "cashUnitError", | string | |
| "countedCashUnits": { | object | |

```

"additionalProperties": {          object
"physicalPositionName": Add example to YAML,  string
"unitId": Add example to YAML,           string
"dispensed": 0,                   integer
"counted": 0,                   integer
"status": "ok"                  string
}
}
}

```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

- cashUnitError - A cash unit caused a problem. A [CashManagement.CashUnitErrorEvent](#) will be posted with the details.
- unsupportedPosition - The position specified is not supported.
- safeDoorOpen - The safe door is open. This device requires the safe door to be closed in order to perform this operation.
- exchangeActive - The device is in an exchange state (see [CashManagement.StartExchange](#)).

countedCashUnits

List of counted cash unit objects.

countedCashUnits/additionalProperties

Counted cash unit object. Object name is the same as used in [CashManagement.GetCashUnitInfo](#).

countedCashUnits/additionalProperties/physicalPositionName

Specifies which cash unit was emptied and counted. This name is the same as the *physicalPositionName* in the [CashManagement.GetCashUnitInfo](#) completion message.

countedCashUnits/additionalProperties/unitId

Cash unit ID. This is the identifier defined in the *unitID* field in the [CashManagement.GetCashUnitInfo](#) completion message.

countedCashUnits/additionalProperties/dispensed

The number of items that were dispensed during the emptying of the cash unit.

countedCashUnits/additionalProperties/collected

The number of items that were counted during the emptying of the cash unit.

countedCashUnits/additionalProperties/status

Supplies the status of the cash unit. Following values are possible:

- ok - The cash unit is in a good state.
- full - The cash unit is full.
- high - The cash unit is almost full (i.e. reached or exceeded the threshold defined by *maximum*).
- low - The cash unit is almost empty (i.e. reached or below the threshold defined by *minimum*).
- empty - The cash unit is empty, or insufficient items in the cash unit are preventing further dispense operations.
- inoperative - The cash unit is inoperative.
- missing - The cash unit is missing.
- noValue - The values of the specified cash unit are not available.
- noReference - There is no reference value available for the notes in this cash unit. The cash unit has not been calibrated.

- manipulated - The cash unit has been inserted (including removal followed by a reinsertion) when the device was not in the exchange state. This cash unit cannot be dispensed from.

Event Messages

- [CashManagement.CashUnitErrorEvent](#)
- [Dispenser.ItemsTakenEvent](#)
- [Dispenser.ItemsPresentedEvent](#)
- [CashManagement.NoteErrorEvent](#)
- [CashManagement.InfoAvailableEvent](#)
- [Dispenser.ShutterStatusChangedEvent](#)

[5.2.15 - Dispenser.PrepareDispense](#)

On some hardware it can take a significant amount of time for the dispenser to get ready to dispense media. On this type of hardware the Dispenser.PrepareDispense command can be used to improve transaction performance.

If this command is supported (see the *prepareDispense* capability) then clients can help to improve the time taken to dispense media by issuing this command as soon as the client knows that a dispense is likely to happen. This command either prepares the device for the next dispense operation, or terminates the dispense preparation if the subsequent dispense operation is no longer required.

With the exception of the Dispenser.Denominate and Dispenser.Dispense commands, which will not stop the dispense preparation, any execute command on Dispenser or CashAcceptor will automatically stop the dispense preparation.

If this command is executed and the device is already in the specified *action* state, then this execution will have no effect and will complete with a successful completion message.

Command Message

| Payload | Type | Required |
|--------------------------|------|----------|
| { | | |
| "timeout": 5000, integer | | |
| "action": "start" string | | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

action

A value specifying the type of actions. Following values are possible:

- start - Initiates the action to prepare for the next dispense command. This command does not wait until the device is ready to dispense before returning a completion event, it completes as soon as the preparation has been initiated.
- stop - Stops the previously activated dispense preparation. For example the motor of the transport will be stopped. This should be used if for some reason the subsequent dispense operation is no longer required.

Completion Message

| Payload | Type | Required |
|---|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

Event Messages

None

5.3 - Event Messages

5.3.1 - CashManagement.CashUnitErrorEvent

This event is generated if there is a problem with a cash unit during the execution of a command.

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "failure": "empty", | string | |
| "cashUnit": { | object | |
| "status": "ok", | string | |
| "type": "billCassette", | string | |
| "currencyID": Add example to YAML, | string | |
| "value": 0, | number | |
| "logicalCount": 0, | integer | |
| "maximum": 0, | integer | |
| "appLock": false, | boolean | |
| "cashUnitName": Add example to YAML, | string | |
| "initialCount": 0, | integer | |
| "dispensedCount": 0, | integer | |
| "presentedCount": 0, | integer | |
| "retractedCount": 0, | integer | |
| "rejectCount": 0, | integer | |
| "minimum": 0, | integer | |
| "physicalPositionName": Add example to YAML, | string | |

```

"unitID": Add example to YAML,           string
"count": 0,                            integer
"maximumCapacity": 0,                  integer
"hardwareSensor": false,              boolean
"itemType": {                           object
  "all": false,                         boolean
  "unfit": false,                        boolean
  "individual": false,                   boolean
  "level1": false,                       boolean
  "level2": false,                       boolean
  "level3": false,                       boolean
  "itemProcessor": false,                 boolean
  "unfitIndividual": false             boolean
},
"cashInCount": 0,                      integer
"noteNumberList": {                    object
  "noteNumber": [{                     array (object)
    "noteID": 0,                         integer
    "count": 0                           integer
  }]
},
"noteIDs": [0]                          array (integer)
}
}

```

Properties

failure

Specifies the kind of failure that occurred in the cash unit. Following values are possible:

- empty - Specified cash unit is empty.
- error - Specified cash unit has malfunctioned.
- full - Specified cash unit is full.
- locked - Specified cash unit is locked.
- invalid - Specified cash unit is invalid.
- config - An attempt has been made to change the settings of a self-configuring cash unit.
- notConfigured - Specified cash unit is not configured.

cashUnit

The cash unit object that caused the problem.

cashUnit/status

Supplies the status of the cash unit. Following values are possible:

- ok - The cash unit is in a good state.
- full - The cash unit is full.
- high - The cash unit is almost full (i.e. reached or exceeded the threshold defined by *maximum*).
- low - The cash unit is almost empty (i.e. reached or below the threshold defined by *minimum*).
- empty - The cash unit is empty, or insufficient items in the cash unit are preventing further dispense operations.
- inoperative - The cash unit is inoperative.
- missing - The cash unit is missing.
- noValue - The values of the specified cash unit are not available.
- noReference - There is no reference value available for the notes in this cash unit. The cash unit has not been calibrated.
- manipulated - The cash unit has been inserted (including removal followed by a reinsertion) when the device was not in the exchange state. This cash unit cannot be dispensed from.

cashUnit/type

Type of cash unit. Following values are possible:

- notApplicable - Not applicable. Typically means cash unit is missing.
- rejectCassette - Reject cash unit. This type will also indicate a combined

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

reject/retract cash unit.

- billCassette - Cash unit containing bills.
- coinCylinder - Coin cylinder.
- coinDispenser - Coin dispenser as a whole unit.
- retractCassette - Retract cash unit.
- coupon - Cash unit containing coupons or advertising material.
- document - Cash unit containing documents.
- replenishmentContainer - Replenishment container. A cash unit can be refilled from a replenishment container.
- recycling - Recycling cash unit. This unit is only present when the device implements the Dispenser and CashAcceptor interfaces.
- cashIn - Cash-in cash unit.

cashUnit/currencyID

A three character string storing the ISO format [Ref. 2] Currency ID. This value will be omitted for cash units which contain items of more than one currency type or items to which currency is not applicable. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

cashUnit/value

Supplies the value of a single item in the cash unit. This value is expressed as floating point value. If the *currencyID* field for this cash unit is omitted, then this field will contain zero. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

cashUnit/logicalCount

The meaning of this count depends on the type of cash unit. This value is persistent. For all cash units except retract cash units (*type* is not *retractCassette*) this value specifies the number of items inside the cash unit. For all dispensing cash units (*type* is *billCassette*, *coinCylinder*, *coinDispenser*, *coupon*, *document* or *recycling*), this value includes any items from the cash unit not yet presented to the customer. This count is only decremented when the items are either known to be in customer access or successfully rejected. If the cash unit is usable from the CashAcceptor interface (*type* is *recycling*, *cashIn*, *retractCassette* or *rejectCassette*) then this value will be incremented as a result of a cash-in operation. Note that for a reject cash unit (*type* is *rejectCassette*), this value is unreliable, since the typical reason for dumping items to the reject cash unit is a suspected count failure. For a retract cash unit (*type* is *retractCassette*) this value specifies the number of retract operations which result in items entering the cash unit.

cashUnit/maximum

When *count* reaches this value the threshold event

CashManagement.CashUnitThresholdEvent (*high*) will be generated. This value can be different from the actual capacity of the cassette. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

cashUnit/appLock

If this value is TRUE items cannot be dispensed from or deposited into the cash unit. If this value is TRUE and the client attempts to use the cash unit a CashManagement.CashUnitErrorEvent event will be generated and an error completion message will be returned. This value is persistent.

cashUnit/cashUnitName

A name which helps to identify the type of the cash unit. This is especially useful in the case of cash units of type *document* where different documents can have the same currency and value. For example, travelers checks and bank checks may have the same currency and value but still need to be identifiable as different types of document. Where this value is not relevant (e.g. in bill cash units) the property can be omitted. This value is persistent.

cashUnit/initialCount

Initial number of items contained in the cash unit. This value is persistent.

cashUnit/dispensedCount

The number of items dispensed from this cash unit. This count is incremented when the items are removed from the cash units. This count includes any items that were rejected during the dispense operation and are no longer in this cash unit. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

cashUnit/presentedCount

The number of items from this cash unit that have been presented to the customer. This count is incremented when the items are presented to the customer. If it is unknown if a customer has been presented with the items, then this count is not updated. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

cashUnit/retractedCount

The number of items that have been accessible to a customer and retracted into the cash unit. This value is persistent.

cashUnit/rejectCount

The number of items dispensed from this cash unit which have been rejected, are in a cash unit other than this cash unit, and which have not been accessible to a customer. This value may be unreliable, since a typical reason for rejecting items is a suspected pick failure. Other reasons for rejecting items may include incorrect note denominations, classifications not valid for dispensing, or where the transaction has been cancelled and a Reject command has been called. For reject and retract cash units (*type* is *rejectCassette* or *retractCassette*) this field does not apply and will be reported as zero. This value is persistent.

cashUnit/minimum

This field is not applicable to retract and reject cash units. For all cash units which dispense items (all other), when *count* reaches this value the threshold event CashManagement.CashUnitThresholdEvent (*low*) will be generated. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

cashUnit/physicalPositionName

A name identifying the physical location of the cash unit.

cashUnit/unitID

A 5 character string uniquely identifying the cash unit.

cashUnit/count

As defined by the *logicalCount* description, but with the following exceptions: This count does not include items dispensed but not yet presented. On cash units with *type* set to "retractCassette" the count represents the number of items, unless the device cannot count items during a retract, in which case this count will be zero. This value is persistent.

cashUnit/maximumCapacity

The maximum number of items the cash unit can hold. This is only for informational purposes. No threshold event CashManagement.CashUnitThresholdEvent will be generated. This value is persistent.

cashUnit/hardwareSensor

Specifies whether or not threshold events can be generated based on hardware sensors in the device. If this value is TRUE then threshold events may be generated based on hardware sensors as opposed to counts.

cashUnit/itemType

Specifies the type of items the cash unit takes as a combination of the following flags. The table in the Comments section of this command defines how to interpret the combination of these flags (TODO: include Table)

cashUnit/itemType/all

The cash unit takes all fit banknote types. These are level 4 notes which are fit for recycling.

cashUnit/itemType/unfit

The cash unit takes all unfit banknotes. These are level 4 notes which are unfit for recycling.

cashUnit/itemType/individual

The cash unit takes all types of fit banknotes specified in an individual list. These are level 4 notes which are fit for recycling.

cashUnit/itemType/level1

Level 1 note types are stored in this cash unit.

cashUnit/itemType/level2

If notes can be classified as level 2, then level 2 note types are stored in this cash unit.

cashUnit/itemType/level3

If notes can be classified as level 3, then level 3 note types are stored in this cash unit.

cashUnit/itemType/itemProcessor

The cash unit can accept items on the ItemProcessor interface.

cashUnit/itemType/unfitIndividual

The cash unit takes all types of unfit banknotes specified in an individual list. These are level 4 notes which are unfit for recycling.

cashUnit/cashInCount

Count of items that have entered the cash unit. This counter is incremented whenever an item enters a cash unit for any reason, unless it originated from this cash unit but was returned without being accessible to a customer. For a retract cash unit this value represents the total number of items of all types in the cash unit, or if the device cannot count items during a retract operation this value will be zero. This value is persistent.

cashUnit/noteNumberList

Array of cash items inside the cash unit. The content of this structure is persistent. If the cash unit is Dispenser specific cash unit with *type billCassette* or the contents of the cash unit are not known this structure will be omitted. If the cash unit is of type *retractCassette* this pointer will be omitted except for the following cases:

- If the retract cash unit is configured to accept level 2 notes then the number and type of level 2 notes is returned in the *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of level 2 notes.
- If items are recognized during retract operations then the number and type of notes retracted is returned in *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of retracted items.

cashUnit/noteNumberList/noteNumber

Array of banknote numbers the cash unit contains.

cashUnit/noteNumberList/noteNumber/noteID

Identification of note type. The Note ID represents the note identifiers reported by the `CashAcceptor.BanknoteTypes` command. If this value is zero then the note type is unknown.

`cashUnit/noteNumberList/noteNumber/count`

Actual count of cash items. The value is incremented each time cash items are moved to a cash unit. In the case of recycle cash units this count is decremented as defined in the description of the `logicalCount` field.

`cashUnit/noteIDs`

Array of integers which contains the note IDs of the banknotes the cash-in cash unit or recycle cash unit can take. This field only applies to *individual* cassette types. If there are no note IDs defined for the cassette or the cassette is not defined as *individual* then `noteIDs` will be omitted.

5.3.2 - `CashManagement.CashUnitThresholdEvent`

This user event is generated when a threshold condition has occurred in one of the cash units.

This event can be triggered either by hardware sensors in the device or by the `count` reaching the `minimum` or `maximum` value as specified in the `GetCashUnitInfo` structure.

The client can check if the device has hardware sensors by querying the `hardwareSensor` field of the cash unit structure. If a cash unit has this capability then threshold events based on hardware sensors will be triggered if the `maximum` or `minimum` values are not used or are set to zero.

| Payload | Type | Required |
|---|---------|----------|
| { | | |
| <code>"status": "ok",</code> | string | |
| <code>"type": "billCassette",</code> | string | |
| <code>"currencyID": Add example to YAML,</code> | string | |
| <code>"value": 0,</code> | number | |
| <code>"logicalCount": 0,</code> | integer | |
| <code>"maximum": 0,</code> | integer | |
| <code>"appLock": false,</code> | boolean | |

```

"cashUnitName": Add example to YAML,           string
"initialCount": 0,                            integer
"dispensedCount": 0,                          integer
"presentedCount": 0,                          integer
"retractedCount": 0,                          integer
"rejectCount": 0,                            integer
"minimum": 0,                                integer
"physicalPositionName": Add example to YAML, string
"unitID": Add example to YAML,                string
"count": 0,                                  integer
"maximumCapacity": 0,                         integer
"hardwareSensor": false,                      boolean
"itemType": {                                 object
  "all": false,                             boolean
  "unfit": false,                           boolean
  "individual": false,                     boolean
  "level1": false,                          boolean
  "level2": false,                          boolean
  "level3": false,                          boolean
  "itemProcessor": false,                   boolean
  "unfitIndividual": false,                 boolean
},
"cashInCount": 0,                            integer
"noteNumberList": {                           object
  "noteNumber": [{                           array (object)
    "noteID": 0,                            integer
  }
]
}

```

```

"count": 0           integer
}
},
"noteIDs": [0]       array (integer)
}

```

Properties

status

Supplies the status of the cash unit. Following values are possible:

- ok - The cash unit is in a good state.
- full - The cash unit is full.
- high - The cash unit is almost full (i.e. reached or exceeded the threshold defined by *maximum*).
- low - The cash unit is almost empty (i.e. reached or below the threshold defined by *minimum*).
- empty - The cash unit is empty, or insufficient items in the cash unit are preventing further dispense operations.
- inoperative - The cash unit is inoperative.
- missing - The cash unit is missing.
- noValue - The values of the specified cash unit are not available.
- noReference - There is no reference value available for the notes in this cash unit. The cash unit has not been calibrated.
- manipulated - The cash unit has been inserted (including removal followed by a reinsertion) when the device was not in the exchange state. This cash unit cannot be dispensed from.

type

Type of cash unit. Following values are possible:

- notApplicable - Not applicable. Typically means cash unit is missing.
- rejectCassette - Reject cash unit. This type will also indicate a combined reject/retract cash unit.
- billCassette - Cash unit containing bills.
- coinCylinder - Coin cylinder.
- coinDispenser - Coin dispenser as a whole unit.
- retractCassette - Retract cash unit.

- coupon - Cash unit containing coupons or advertising material.
- document - Cash unit containing documents.
- replenishmentContainer - Replenishment container. A cash unit can be refilled from a replenishment container.
- recycling - Recycling cash unit. This unit is only present when the device implements the Dispenser and CashAcceptor interfaces.
- cashIn - Cash-in cash unit.

currencyID

A three character string storing the ISO format [Ref. 2] Currency ID. This value will be omitted for cash units which contain items of more than one currency type or items to which currency is not applicable. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

value

Supplies the value of a single item in the cash unit. This value is expressed as floating point value. If the *currencyID* field for this cash unit is omitted, then this field will contain zero. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

logicalCount

The meaning of this count depends on the type of cash unit. This value is persistent. For all cash units except retract cash units (*type* is not *retractCassette*) this value specifies the number of items inside the cash unit. For all dispensing cash units (*type* is *billCassette*, *coinCylinder*, *coinDispenser*, *coupon*, *document* or *recycling*), this value includes any items from the cash unit not yet presented to the customer. This count is only decremented when the items are either known to be in customer access or successfully rejected. If the cash unit is usable from the CashAcceptor interface (*type* is *recycling*, *cashIn*, *retractCassette* or *rejectCassette*) then this value will be incremented as a result of a cash-in operation. Note that for a reject cash unit (*type* is *rejectCassette*), this value is unreliable, since the typical reason for dumping items to the reject cash unit is a suspected count failure. For a retract cash unit (*type* is *retractCassette*) this value specifies the number of retract operations which result in items entering the cash unit.

maximum

When *count* reaches this value the threshold event

CashManagement.CashUnitThresholdEvent (*high*) will be generated. This value can be different from the actual capacity of the cassette. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware

sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

appLock

If this value is TRUE items cannot be dispensed from or deposited into the cash unit. If this value is TRUE and the client attempts to use the cash unit a *CashManagement.CashUnitErrorEvent* event will be generated and an error completion message will be returned. This value is persistent.

cashUnitName

A name which helps to identify the type of the cash unit. This is especially useful in the case of cash units of type *document* where different documents can have the same currency and value. For example, travelers checks and bank checks may have the same currency and value but still need to be identifiable as different types of document. Where this value is not relevant (e.g. in bill cash units) the property can be omitted. This value is persistent.

initialCount

Initial number of items contained in the cash unit. This value is persistent.

dispensedCount

The number of items dispensed from this cash unit. This count is incremented when the items are removed from the cash units. This count includes any items that were rejected during the dispense operation and are no longer in this cash unit. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

presentedCount

The number of items from this cash unit that have been presented to the customer. This count is incremented when the items are presented to the customer. If it is unknown if a customer has been presented with the items, then this count is not updated. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

retractedCount

The number of items that have been accessible to a customer and retracted into the cash unit. This value is persistent.

rejectCount

The number of items dispensed from this cash unit which have been rejected, are in a cash unit other than this cash unit, and which have not been accessible to a customer. This value may be unreliable, since a typical reason for rejecting items is a suspected pick failure. Other reasons for rejecting items may include incorrect note denominations, classifications not valid for dispensing, or where the transaction has been cancelled and a Reject command has been called. For reject and retract cash units (*type* is *rejectCassette* or *retractCassette*) this field does not apply and will be reported as zero. This value is persistent.

minimum

This field is not applicable to retract and reject cash units. For all cash units which dispense items (all other), when *count* reaches this value the threshold event CashManagement.CashUnitThresholdEvent (*low*) will be generated. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

physicalPositionName

A name identifying the physical location of the cash unit.

unitID

A 5 character string uniquely identifying the cash unit.

count

As defined by the *logicalCount* description, but with the following exceptions: This count does not include items dispensed but not yet presented. On cash units with *type* set to "retractCassette" the count represents the number of items, unless the device cannot count items during a retract, in which case this count will be zero. This value is persistent.

maximumCapacity

The maximum number of items the cash unit can hold. This is only for informational purposes. No threshold event CashManagement.CashUnitThresholdEvent will be generated. This value is persistent.

hardwareSensor

Specifies whether or not threshold events can be generated based on hardware sensors in the device. If this value is TRUE then threshold events may be generated based on hardware sensors as opposed to counts.

itemType

Specifies the type of items the cash unit takes as a combination of the following flags. The table in the Comments section of this command defines how to interpret the combination of these flags (TODO: include Table)

itemType/all

The cash unit takes all fit banknote types. These are level 4 notes which are fit for recycling.

itemType/unfit

The cash unit takes all unfit banknotes. These are level 4 notes which are unfit for recycling.

itemType/individual

The cash unit takes all types of fit banknotes specified in an individual list. These are level 4 notes which are fit for recycling.

itemType/level1

Level 1 note types are stored in this cash unit.

itemType/level2

If notes can be classified as level 2, then level 2 note types are stored in this cash unit.

itemType/level3

If notes can be classified as level 3, then level 3 note types are stored in this cash unit.

itemType/itemProcessor

The cash unit can accept items on the ItemProcessor interface.

itemType/unfitIndividual

The cash unit takes all types of unfit banknotes specified in an individual list. These are level 4 notes which are unfit for recycling.

cashInCount

Count of items that have entered the cash unit. This counter is incremented whenever an item enters a cash unit for any reason, unless it originated from this cash unit but was returned without being accessible to a customer. For a retract cash unit this value represents the total number of items of all types in the cash unit, or if the device cannot count items during a retract operation this value will be zero. This value is persistent.

noteNumberList

Array of cash items inside the cash unit. The content of this structure is persistent. If the cash unit is Dispenser specific cash unit with *type billCassette* or the contents of the cash unit are not known this structure will be omitted. If the cash unit is of *type retractCassette* this pointer will be omitted except for the following cases:

- If the retract cash unit is configured to accept level 2 notes then the number and type of level 2 notes is returned in the *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of level 2 notes.
- If items are recognized during retract operations then the number and type of notes retracted is returned in *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of retracted items.

noteNumberList/noteNumber

Array of banknote numbers the cash unit contains.

noteNumberList/noteNumber/noteID

Identification of note type. The Note ID represents the note identifiers reported by the *CashAcceptor.BanknoteTypes* command. If this value is zero then the note type is unknown.

noteNumberList/noteNumber/count

Actual count of cash items. The value is incremented each time cash items are moved to a

cash unit. In the case of recycle cash units this count is decremented as defined in the description of the *logicalCount* field.

noteIDs

Array of integers which contains the note IDs of the banknotes the cash-in cash unit or recycle cash unit can take. This field only applies to *individual* cassette types. If there are no note IDs defined for the cassette or the cassette is not defined as *individual* then *noteIDs* will be omitted.

[5.3.3 - Dispenser.DelayedDispenseEvent](#)

This event is generated if the start of a dispense operation has been delayed.

| Payload | Type | Required |
|------------|---------|----------|
| { | | |
| "delay": 0 | integer | |
| } | | |

Properties

delay

The time in milliseconds by which the dispense operation will be delayed.

[5.3.4 - Dispenser.StartDispenseEvent](#)

This event is generated when a delayed dispense operation begins.

| Payload | Type | Required |
|------------|---------|----------|
| { | | |
| "reqID": 0 | integer | |
| } | | |

Properties

reqID

The requestId of the original dispense command.

5.3.5 - Dispenser.PartialDispenseEvent

This event is generated when a dispense operation is divided into several sub-dispense operations because the hardware capacity of the Dispenser is exceeded.

| Payload | Type | Required |
|--------------|---------|----------|
| { | | |
| "dispNum": 0 | integer | |
| } | | |

Properties**dispNum**

The number of sub-dispense operations into which the dispense operation has been divided.

5.3.6 - Dispenser.SubDispenseOkEvent

This event is generated when one of the sub-dispense operations into which the dispense operation was divided has finished successfully. To present the items to the customer a Dispenser.Present command has to be issued. Note that in this case the values in the payload structure report the amount and number of each denomination dispensed in the sub-dispense operation.

| Payload | Type | Required |
|---------------------------|--------|----------|
| { | | |
| "currencies": { | object | |
| "additionalProperties": 0 | number | |
| }, | | |
| "values": { | object | |

```
"additionalProperties": 0 integer  
},  
"cashBox": 0 integer  
}
```

Properties

currencies

"List of currency and amount combinations for denomination. There will be one entry for each currency in the denomination. The property name is the currency name in ISO format (e.g. "EUR").

currencies/additionalProperties

The amount to be denominated or dispensed. Use currency identifier in ISO format as property name.

values

This list specifies the number of items to take from the cash units. Each entry uses a cashunit object name as stated by the [CashManagement.GetCashUnitInfo](#) command. The value of the entry is the number of items to take from that unit. If the client does not wish to specify a denomination, it should omit the values property.

values/additionalProperties

Number of items to take from the specified cash unit. Use cash unit name as specified by the [CashManagement.GetCashUnitInfo](#) command as property name.

cashBox

Only applies to Teller Dispensers. Amount to be paid from the teller's cash box.

5.3.7 - Dispenser.IncompleteDispenseEvent

This event is generated during Dispenser.Dispense when it has not been possible to dispense the entire denomination but part of the requested denomination is on the intermediate stacker or in customer access. Note that in this case the values in this structure report the amount and number of each denomination that are in customer access or on the intermediate stacker. Dispenser.GetPresentStatus can be used to determine whether the items are in customer access.

| Payload | Type | Required |
|---------------------------|---------|----------|
| { | | |
| "currencies": { | object | |
| "additionalProperties": 0 | number | |
| }, | | |
| "values": { | object | |
| "additionalProperties": 0 | integer | |
| }, | | |
| "cashBox": 0 | integer | |
| } | | |

Properties

currencies

"List of currency and amount combinations for denomination. There will be one entry for each currency in the denomination. The property name is the currency name in ISO format (e.g. "EUR").

currencies/additionalProperties

The amount to be denominated or dispensed. Use currency identifier in ISO format as property name.

values

This list specifies the number of items to take from the cash units. Each entry uses a cashunit object name as stated by the [CashManagement.GetCashUnitInfo](#) command. The

value of the entry is the number of items to take from that unit. If the client does not wish to specify a denomination, it should omit the values property.

values/additionalProperties

Number of items to take from the specified cash unit. Use cash unit name as specified by the [CashManagement.GetCashUnitInfo](#) command as property name.

cashBox

Only applies to Teller Dispensers. Amount to be paid from the teller's cash box.

5.3.8 - [CashManagement.NoteErrorEvent](#)

This event specifies the reason for a note detection error during the execution of a command.

| Payload | Type | Required |
|--|------|----------|
| { "reason": "doubleNote" string } | | |

Properties

reason

The reason for the notes detection error. Following values are possible:

- doubleNote - Double notes have been detected.
- longNote - A long note has been detected.
- skewedNote - A skewed note has been detected.
- incorrectCount - An item counting error has occurred.
- notesTooClose - Notes have been detected as being too close.
- otherNoteError - An item error not covered by the other values has been detected.
- shortNote - Short notes have been detected.

5.3.9 - CashManagement.InfoAvailableEvent

This execute event is generated when information is available for items detected during the cash processing operation.

| Payload | Type | Required |
|-----------------------|----------------|----------|
| { | | |
| "itemInfoSummary": [{ | array (object) | |
| "level": "level1", | string | |
| "numOfItems": 0 | integer | |
| }] | | |
| } | | |

Properties

itemInfoSummary

Array of itemInfoSummary objects, one object for every level.

itemInfoSummary/level

Defines the note level. Following values are possible:

- level1 - Information for level 1 notes.
- level2 - Information for level 2 notes.
- level3 - Information for level 3 notes.
- level4 - Information for level 4 notes.

itemInfoSummary/numOfItems

Number of items classified as *level* which have information available.

5.3.10 - Dispenser.IncompleteRetractEvent

This event is sent when a retract or reset command has completed with an error and not all of the items have been retracted.

| Payload | Type | Required |
|---------|------|----------|
|---------|------|----------|

```
{
  "itemNumberList": { object
    "itemNumber": [ array (object)
      "currencyID": Add example to YAML, string
      "values": 0, number
      "release": 0, integer
      "count": 0, integer
      "cashunit": Add example to YAML string
    ]
  },
  "reason": "retractFailure" string
}
```

Properties

itemNumberList

The values in this structure report the amount and number of each denomination that were successfully moved during the command prior to the failure.

itemNumberList/itemNumber

Array of item number objects.

itemNumberList/itemNumber/currencyID

A three character array storing the ISO format [Ref. 2] Currency ID; if the currency of the item is not known this is omitted.

itemNumberList/itemNumber/values

The value of a single item expressed as floating point value; or a zero value if the value of the item is not known.

itemNumberList/itemNumber/release

The release of the item. The higher this number is, the newer the release. Zero means that there is only one release or the release is not known. This value has not been standardized and therefore a release number of the same item will not necessarily have the same value in different systems.

itemNumberList/itemNumber/count

The count of items of the same type moved to the same destination during the execution of this command.

itemNumberList/itemNumber/cashunit

The object name of the cash unit which received items during the execution of this command as stated by the [CashManagement.GetCashUnitInfo](#) command. This value will be omitted if items were moved to the [retractArea](#) transport or stacker.

reason

The reason for not having retracted items. Following values are possible:

- retractFailure - The retract has partially failed for a reason not covered by the other reasons listed in this event, for example failing to pick an item to be retracted.
- retractAreaFull - The specified retract area (see input parameter *retractArea*) has become full during the retract operation.
- foreignItemsDetected - Foreign items have been detected.
- invalidBunch - An invalid bunch of items has been detected, e.g. it is too large or could not be processed.

[5.3.11 - CashManagement.CashUnitInfoChangedEvent](#)

This service event is generated under the following circumstances:

- It is generated whenever *status* changes. For instance, a cash unit has been removed or inserted, or a cash unit has become empty or full.
- This event will also be generated for every cash unit changed in any way (including changes to counts, e.g. *count*, *rejectCount*, *initialCount*, *dispensedCount* and *presentedCount*) as a result of the [CashManagement.SetCashUnitInfo](#) command.

- This event will also be fired when any change is made to a cash unit by the following commands, except for changes to counts (e.g. *count*, *rejectCount*, *initialCount*, *dispensedCount* and *presentedCount*):

[Dispenser.CalibrateCashUnit](#)

[Dispenser.TestCashUnit](#)

- In addition this event will be generated when a cash unit has been counted during the [CashAcceptor.CashUnitCount](#) command execution.

When a cash unit is removed, the status of the cash unit becomes missing.

If a new cash unit is inserted the cash unit structure reported by the last [CashManagement.GetCashUnitInfo](#) command is no longer valid. In that case a client should issue a [CashManagement.GetCashUnitInfo](#) command after receiving this event to obtain updated cash unit information.

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "status": "ok", | string | |
| "type": "billCassette", | string | |
| "currencyID": Add example to YAML, | string | |
| "value": 0, | number | |
| "logicalCount": 0, | integer | |
| "maximum": 0, | integer | |
| "appLock": false, | boolean | |
| "cashUnitName": Add example to YAML, | string | |
| "initialCount": 0, | integer | |
| "dispensedCount": 0, | integer | |
| "presentedCount": 0, | integer | |
| "retractedCount": 0, | integer | |
| "rejectCount": 0, | integer | |
| "minimum": 0, | integer | |
| "physicalPositionName": Add example to YAML, | string | |

| | |
|--------------------------------|-----------------|
| "unitID": Add example to YAML, | string |
| "count": 0, | integer |
| "maximumCapacity": 0, | integer |
| "hardwareSensor": false, | boolean |
| "itemType": { | object |
| "all": false, | boolean |
| "unfit": false, | boolean |
| "individual": false, | boolean |
| "level1": false, | boolean |
| "level2": false, | boolean |
| "level3": false, | boolean |
| "itemProcessor": false, | boolean |
| "unfitIndividual": false | boolean |
| } | |
| "cashInCount": 0, | integer |
| "noteNumberList": { | object |
| "noteNumber": [{ | array (object) |
| "noteID": 0, | integer |
| "count": 0 | integer |
| } | |
| }, | |
| "noteIDs": [0] | array (integer) |
| } | |

Properties

status

Supplies the status of the cash unit. Following values are possible:

- ok - The cash unit is in a good state.
- full - The cash unit is full.
- high - The cash unit is almost full (i.e. reached or exceeded the threshold defined by *maximum*).
- low - The cash unit is almost empty (i.e. reached or below the threshold defined by *minimum*).
- empty - The cash unit is empty, or insufficient items in the cash unit are preventing further dispense operations.
- inoperative - The cash unit is inoperative.
- missing - The cash unit is missing.
- noValue - The values of the specified cash unit are not available.
- noReference - There is no reference value available for the notes in this cash unit. The cash unit has not been calibrated.
- manipulated - The cash unit has been inserted (including removal followed by a reinsertion) when the device was not in the exchange state. This cash unit cannot be dispensed from.

type

Type of cash unit. Following values are possible:

- notApplicable - Not applicable. Typically means cash unit is missing.
- rejectCassette - Reject cash unit. This type will also indicate a combined reject/retract cash unit.
- billCassette - Cash unit containing bills.
- coinCylinder - Coin cylinder.
- coinDispenser - Coin dispenser as a whole unit.
- retractCassette - Retract cash unit.
- coupon - Cash unit containing coupons or advertising material.
- document - Cash unit containing documents.
- replenishmentContainer - Replenishment container. A cash unit can be refilled from a replenishment container.
- recycling - Recycling cash unit. This unit is only present when the device implements the Dispenser and CashAcceptor interfaces.
- cashIn - Cash-in cash unit.

currencyID

A three character string storing the ISO format [Ref. 2] Currency ID. This value will be omitted for cash units which contain items of more than one currency type or items to which currency is not applicable. If the *status* field for this cash unit is *noValue* it is the

responsibility of the client to assign a value to this field. This value is persistent.

value

Supplies the value of a single item in the cash unit. This value is expressed as floating point value. If the *currencyID* field for this cash unit is omitted, then this field will contain zero. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

logicalCount

The meaning of this count depends on the type of cash unit. This value is persistent. For all cash units except retract cash units (*type* is not *retractCassette*) this value specifies the number of items inside the cash unit. For all dispensing cash units (*type* is *billCassette*, *coinCylinder*, *coinDispenser*, *coupon*, *document* or *recycling*), this value includes any items from the cash unit not yet presented to the customer. This count is only decremented when the items are either known to be in customer access or successfully rejected. If the cash unit is usable from the CashAcceptor interface (*type* is *recycling*, *cashIn*, *retractCassette* or *rejectCassette*) then this value will be incremented as a result of a cash-in operation. Note that for a reject cash unit (*type* is *rejectCassette*), this value is unreliable, since the typical reason for dumping items to the reject cash unit is a suspected count failure. For a retract cash unit (*type* is *retractCassette*) this value specifies the number of retract operations which result in items entering the cash unit.

maximum

When *count* reaches this value the threshold event

CashManagement.CashUnitThresholdEvent (*high*) will be generated. This value can be different from the actual capacity of the cassette. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

appLock

If this value is TRUE items cannot be dispensed from or deposited into the cash unit. If this value is TRUE and the client attempts to use the cash unit a CashManagement.CashUnitErrorEvent event will be generated and an error completion message will be returned. This value is persistent.

cashUnitName

A name which helps to identify the type of the cash unit. This is especially useful in the case

of cash units of type *document* where different documents can have the same currency and value. For example, travelers checks and bank checks may have the same currency and value but still need to be identifiable as different types of document. Where this value is not relevant (e.g. in bill cash units) the property can be omitted. This value is persistent.

initialCount

Initial number of items contained in the cash unit. This value is persistent.

dispensedCount

The number of items dispensed from this cash unit. This count is incremented when the items are removed from the cash units. This count includes any items that were rejected during the dispense operation and are no longer in this cash unit. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

presentedCount

The number of items from this cash unit that have been presented to the customer. This count is incremented when the items are presented to the customer. If it is unknown if a customer has been presented with the items, then this count is not updated. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

retractedCount

The number of items that have been accessible to a customer and retracted into the cash unit. This value is persistent.

rejectCount

The number of items dispensed from this cash unit which have been rejected, are in a cash unit other than this cash unit, and which have not been accessible to a customer. This value may be unreliable, since a typical reason for rejecting items is a suspected pick failure. Other reasons for rejecting items may include incorrect note denominations, classifications not valid for dispensing, or where the transaction has been cancelled and a Reject command has been called. For reject and retract cash units (*type* is *rejectCassette* or *retractCassette*) this field does not apply and will be reported as zero. This value is persistent.

minimum

This field is not applicable to retract and reject cash units. For all cash units which dispense items (all other), when *count* reaches this value the threshold event

CashManagement.CashUnitThresholdEvent (*low*) will be generated. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

physicalPositionName

A name identifying the physical location of the cash unit.

unitID

A 5 character string uniquely identifying the cash unit.

count

As defined by the *logicalCount* description, but with the following exceptions: This count does not include items dispensed but not yet presented. On cash units with *type* set to "retractCassette" the count represents the number of items, unless the device cannot count items during a retract, in which case this count will be zero. This value is persistent.

maximumCapacity

The maximum number of items the cash unit can hold. This is only for informational purposes. No threshold event CashManagement.CashUnitThresholdEvent will be generated. This value is persistent.

hardwareSensor

Specifies whether or not threshold events can be generated based on hardware sensors in the device. If this value is TRUE then threshold events may be generated based on hardware sensors as opposed to counts.

itemType

Specifies the type of items the cash unit takes as a combination of the following flags. The table in the Comments section of this command defines how to interpret the combination of these flags (TODO: include Table)

itemType/all

The cash unit takes all fit banknote types. These are level 4 notes which are fit for recycling.

itemType/unfit

The cash unit takes all unfit banknotes. These are level 4 notes which are unfit for recycling.

itemType/individual

The cash unit takes all types of fit banknotes specified in an individual list. These are level 4 notes which are fit for recycling.

itemType/level1

Level 1 note types are stored in this cash unit.

itemType/level2

If notes can be classified as level 2, then level 2 note types are stored in this cash unit.

itemType/level3

If notes can be classified as level 3, then level 3 note types are stored in this cash unit.

itemType/itemProcessor

The cash unit can accept items on the ItemProcessor interface.

itemType/unfitIndividual

The cash unit takes all types of unfit banknotes specified in an individual list. These are level 4 notes which are unfit for recycling.

cashInCount

Count of items that have entered the cash unit. This counter is incremented whenever an item enters a cash unit for any reason, unless it originated from this cash unit but was

returned without being accessible to a customer. For a retract cash unit this value represents the total number of items of all types in the cash unit, or if the device cannot count items during a retract operation this value will be zero. This value is persistent.

noteNumberList

Array of cash items inside the cash unit. The content of this structure is persistent. If the cash unit is Dispenser specific cash unit with *type billCassette* or the contents of the cash unit are not known this structure will be omitted. If the cash unit is of *type retractCassette* this pointer will be omitted except for the following cases:

- If the retract cash unit is configured to accept level 2 notes then the number and type of level 2 notes is returned in the *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of level 2 notes.
- If items are recognized during retract operations then the number and type of notes retracted is returned in *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of retracted items.

noteNumberList/noteNumber

Array of banknote numbers the cash unit contains.

noteNumberList/noteNumber/noteID

Identification of note type. The Note ID represents the note identifiers reported by the *CashAcceptor.BanknoteTypes* command. If this value is zero then the note type is unknown.

noteNumberList/noteNumber/count

Actual count of cash items. The value is incremented each time cash items are moved to a cash unit. In the case of recycle cash units this count is decremented as defined in the description of the *logicalCount* field.

noteIDs

Array of integers which contains the note IDs of the banknotes the cash-in cash unit or recycle cash unit can take. This field only applies to *individual* cassette types. If there are no note IDs defined for the cassette or the cassette is not defined as *individual* then *noteIDs* will be omitted.

5.4 - Unsolicited Messages

5.4.1 - Dispenser.ItemsTakenEvent

This event is generated when items presented to the user have been taken. This event may be generated at any time.

| Payload | Type | Required |
|-----------------------|--------|----------|
| { | | |
| "position": "default" | string | |
| } | | |

Properties

position

The output position from which the items have been removed. Following values are possible:

- default - The default configuration.
- left - The left output position.
- right - The right output position.
- center - The center output position.
- top - The top output position.
- bottom - The bottom output position.
- front - The front output position.
- rear - The rear output position.

5.4.2 - Dispenser.ItemsPresentedEvent

This event specifies that items have been presented to the user during a count operation and need to be taken.

5.4.3 - Dispenser.MediaDetectedEvent

This service event is generated if media is detected during a reset command. The payload on the event informs the client of the position of the media after the reset completes. If the device has been unable to successfully move the items found then this payload will be omitted.

| Payload | Type | Required |
|----------------------------------|---------|----------|
| { | | |
| "cashunit": Add example to YAML, | string | |
| "retractArea": { | object | |
| "outputPosition": "default", | string | |
| "retractArea": "retract", | string | |
| "index": 0 | integer | |
| }, | | |
| "outputPosition": "default" | string | |
| } | | |

Properties

cashunit

If defined, this value specifies the object name (as stated by the [CashManagement.GetCashUnitInfo](#) command) of the single cash unit to be used for the storage of any items found.

retractArea

This field is used if items are to be moved to internal areas of the device, including cash units, the intermediate stacker, or the transport.

retractArea/outputPosition

Output position from which to retract the items. Following values are possible:

- default - The default configuration information should be used.
- left - Retract items from the left output position.
- right - Retract items from the right output position.
- center - Retract items from the center output position.
- top - Retract items from the top output position.
- bottom - Retract items from the bottom output position.
- front - Retract items from the front output position.
- rear - Retract items from the rear output position.

retractArea/retractArea

This value specifies the area to which the items are to be retracted. Following values are possible:

- retract - Retract the items to a retract cash unit.
- transport - Retract the items to the transport.
- stacker - Retract the items to the intermediate stacker area.
- reject - Retract the items to a reject cash unit.
- itemCassette - Retract the items to the item cassettes, i.e. cassettes that can be dispensed from.

retractArea/index

If *retractArea* is set to "retract" this field defines the position inside the retract cash units into which the cash is to be retracted. *index* starts with a value of one (1) for the first retract position and increments by one for each subsequent position. If there are several retract cash units (of type "retractCassette" in command CashManagement.CashUnitInfo), *index* would be incremented from the first position of the first retract cash unit to the last position of the last retract cash unit. The maximum value of *index* is the sum of *maximum* of each retract cash unit. If *retractArea* is not set to "retract" the value of this field is ignored.

outputPosition

The output position to which items are to be moved. This field is only used if *number* is zero and *retractArea* is omitted. Following values are possible:

- default - The default configuration.
- left - The left output position.
- right - The right output position.
- center - The center output position.
- top - The top output position.
- bottom - The bottom output position.
- front - The front output position.
- rear - The rear output position.

5.4.4 - Dispenser.ShutterStatusChangedEvent

Within the limitations of the hardware sensors this event is generated whenever the status of a shutter changes. The shutter status can change because of an explicit, implicit or manual operation depending on how the shutter is operated.

| Payload | Type | Required |
|---------------------|--------|----------|
| { | | |
| "position": "left", | string | |
| "shutter": "closed" | string | |
| } | | |

Properties

position

Specifies one of the Dispenser output positions whose shutter status has changed. Following values are possible:

- left - Left output position.
- right - Right output position.
- center - Center output position.
- top - Top output position.
- bottom - Bottom output position.
- front - Front output position.
- rear - Rear output position.

shutter

Specifies the new state of the shutter. Following values are possible:

- closed - The shutter is closed.
- open - The shutter is opened.
- jammed - The shutter is jammed.
- unknown - Due to a hardware error or other condition, the state of the shutter cannot be determined.

6 - Cash Acceptor Interface

This chapter defines the Cash Acceptor interface functionality and messages.

6.1 - Summary

This specification describes the functionality of an XFS4IoT compliant Cash Acceptor interface. It defines the service-specific commands that can be issued to the service using the WebSocket endpoint.

Persistent values are maintained through power failures, open sessions, close session and system resets.

This specification covers the acceptance of items. An "item" is defined as any media that can be accepted and includes coupons, documents, bills and coins. However, if coins and bills are both to be accepted separate Services must be implemented for each.

6.2 - Command Messages

6.2.1 - CashAcceptor.GetBanknoteTypes

This command is used to obtain information about the banknote types that can be detected by the banknote reader.

Command Message

| Payload | Type | Required |
|-----------------|---------|----------|
| { | | |
| "timeout": 5000 | integer | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|------------------------------------|----------------|----------|
| { | | |
| "noteTypes": [{ | array (object) | |
| "noteID": 0, | integer | |
| "currencyID": Add example to YAML, | string | |
| "values": 0, | number | |
| "release": 0, | integer | |
| "configured": false | boolean | |
| }] | | |
| } | | |

Properties

noteTypes

List of banknote types the banknote reader supports.

noteTypes/noteID

Identification of note type.

noteTypes/currencyID

Currency ID in ISO 4217 format [Ref. 2].

noteTypes/values

The value of a single item expressed as floating point value.

noteTypes/release

The release of the banknote type. The higher this number is, the newer the release. Zero means that there is only one release of that banknote type. This value has not been standardized and therefore a release number of the same banknote will not necessarily have the same value in different systems.

noteTypes/configured

If TRUE the banknote reader will accept this note type during a cash-in operation, if FALSE the banknote reader will refuse this note type unless it must be retained by note classification rules.

Event Messages

None

6.2.2 - CashAcceptor.GetCashInStatus

This command is used to get information about the status of the currently active cash-in transaction or in the case where no cash-in transaction is active the status of the most recently ended cash-in transaction. This value is persistent and is valid until the next command CashAcceptor.CashInStart.

Command Message

| Payload | Type | Required |
|-----------------|---------|----------|
| { | | |
| "timeout": 5000 | integer | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--------------------------------|----------------|----------|
| { | | |
| "status": "ok", | string | |
| "numOfRefused": 0, | integer | |
| "noteNumberList": { | object | |
| "noteNumber": [{ | array (object) | |
| "noteID": 0, | integer | |
| "count": 0 | integer | |
| }], | | |
| }, | | |
| "unfitNoteNumberList": { | object | |
| See noteNumberList properties. | | |
| } | | |
| } | | |

Properties

status

Status of the currently active or most recently ended cash-in transaction. Following values are possible:

"ok": The cash-in transaction is complete and has ended with a CashAcceptor.CashInEnd command call.

"rollback": The cash-in transaction was has ended with a CashAcceptor.CashInRollback command call.

"active": There is a cash-in transaction active. See the CashAcceptor.CashInStart command description for a definition of an active cash-in transaction.

"retract": The cash-in transaction ended with a Retract command call.

"unknown": The state of the cash-in transaction is unknown. This status is also set if the noteNumberList details are not known or are not reliable.

"reset": The cash-in transaction ended with a Reset command call.

numOfRefused

Specifies the number of items refused during the currently active or most recently ended cash-in transaction period.

noteNumberList

List of banknote types that were inserted, identified and accepted during the currently active or most recently ended cash-in transaction period. If items have been rolled back (status is "rollback") they will be included in this list. If status is "retract" or "reset" then identified and accepted items moved to Cash-In or Recycle cash units are included in this list, but items moved to the Retract or Reject cash units are not included. noteNumberList includes any level 2 or level 3 notes, and all level 4 fit and unfit notes.

noteNumberList/noteNumber

Array of banknote numbers the cash unit contains.

noteNumberList/noteNumber/noteID

Identification of note type. The Note ID represents the note identifiers reported by the *CashAcceptor.BanknoteTypes* command. If this value is zero then the note type is unknown.

noteNumberList/noteNumber/count

Actual count of cash items. The value is incremented each time cash items are moved to a cash unit. In the case of recycle cash units this count is decremented as defined in the description of the *logicalCount* field.

unfitNoteNumberList

List of level 4 unfit banknote types that were inserted, identified and accepted during the currently active or most recently ended cash-in transaction period.

If items have been rolled back (status is "rollback") they will be included in this list. If status is "retract" or "reset" then identified and accepted items moved to Cash-In units are

included in this list, but items moved to the Retract or Reject cash units are not included.

Event Messages

None

6.2.3 - CashAcceptor.GetPositionCapabilities

This command allows the client to get additional information about the use assigned to each position available in the device.

Command Message

| Payload | Type | Required |
|---------------------------|---------|----------|
| { "timeout": 5000 } | integer | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|-----------------------|----------------|----------|
| { | | |
| "posCapabilities": [{ | array (object) | |
| "position": "inLeft", | string | |
| "usage": { | object | |

```

"in": false,           boolean
"refuse": false,      boolean
"rollback": false,    boolean
},
"shutterControl": false, boolean
"itemsTakenSensor": false, boolean
"itemsInsertedSensor": false, boolean
"retractAreas": {       object
  "retract": false,     boolean
  "reject": false,      boolean
  "transport": false,   boolean
  "stacker": false,     boolean
  "billCassettes": false, boolean
  "cashIn": false,      boolean
},
"presentControl": false, boolean
"preparePresent": false, boolean
}]
}

```

Properties

posCapabilities

Array of position capabilities for all positions configured in this service.

posCapabilities/position

Specifies one of the input or output positions as one of the following values:

"inLeft": Left input position.

"inRight": Right input position.

"inCenter": Center input position.

"inTop": Top input position.

"inBottom": Bottom input position.

"inFront": Front input position.

"inRear": Rear input position.

"outLeft": Left output position.

"outRight": Right output position.

"outCenter": Center output position.

"outTop": Top output position.

"outBottom": Bottom output position.

"outFront": Front output position.

"outRear": Rear output position.

posCapabilities/usage

Indicates if an output position is used to reject or rollback.

posCapabilities/usage/in

It is an input position.

posCapabilities/usage/refuse

It is an output position used to refuse items.

posCapabilities/usage/rollback

It is an output position used to rollback items.

posCapabilities/shutterControl

If set to TRUE the shutter is controlled implicitly by the Service. If set to FALSE the shutter must be controlled explicitly by the client using the OpenShutter and the CloseShutter commands. In either case the CashAcceptor.PresentMedia command may be used if the presentControl field is reported as FALSE. The shutterControl field is always set to TRUE if the described position has no shutter.

posCapabilities/itemsTakenSensor

Specifies whether or not the described position can detect when items at the exit position are taken by the user. If set to TRUE the Service generates an accompanying CashAcceptor.ItemsTaken event. If set to FALSE this event is not generated. This field relates to output and refused positions.

posCapabilities/itemsInsertedSensor

Specifies whether the described position has the ability to detect when items have been inserted by the user. If set to TRUE the Service Provider generates an accompanying CashAcceptor.ItemsInserted event. If set to FALSE this event is not generated. This field relates to all input positions.

posCapabilities/retractAreas

Specifies the areas to which items may be retracted from this position. If the device does not have a retract capability all values will be FALSE.

posCapabilities/retractAreas/retract

Items may be retracted to a retract cash unit.

posCapabilities/retractAreas/reject

Items may be retracted to a reject cash unit.

posCapabilities/retractAreas/transport

Items may be retracted to the transport.

posCapabilities/retractAreas/stacker

Items may be retracted to the intermediate stacker.

posCapabilities/retractAreas/billCassettes

Items may be retracted to item cassettes, i.e. cash-in and recycle cash units.

posCapabilities/retractAreas/cashIn

Items may be retracted to a cash-in cash unit.

posCapabilities/presentControl

Specifies how the presenting of media items is controlled. If presentControl is TRUE then the CashAcceptor.PresentMedia command is not supported and items are moved to the output position for removal as part of the relevant command, e.g. CashAcceptor.CashIn or CashAcceptor.CashInRollback where there is implicit shutter control. If presentControl is FALSE then items returned or rejected can be moved to the output position using the CashAcceptor.PresentMedia command, this includes items returned or rejected as part of a CashAcceptor.CashIn or CashAcceptor.CashInRollback operation. The CashAcceptor.PresentMedia command will open and close the shutter implicitly.

posCapabilities/preparePresent

Specifies how the presenting of items is controlled. If preparePresent is FALSE then items to be removed are moved to the output position as part of the relevant command e.g. CashAcceptor.OpenShutter or CashAcceptor.PresentMedia or CashAcceptor.CashInRollback. If preparePresent is TRUE then items are moved to the output position using the CashAcceptor.PreparePresent command.

Event Messages

None

[6.2.4 - CashAcceptor.GetReplenishTarget](#)

This command is used to determine which cash units can be specified as target cash units for a given source cash unit with the CashAcceptor.Replenish command. For example it can be used to determine which targets can be used for replenishment from a replenishment container or from a recycle cash unit.

Command Message

| Payload | Type | Required |
|---------------------------------------|--------|----------|
| { | | |
| "cashunitSource": Add example to YAML | string | |
| } | | |

Properties**cashunitSource**

Object name of the cash unit (as stated by the [CashManagement.GetCashUnitInfo](#) command) which would be used as the source of the replenishment operation.

Completion Message

| Payload | Type | Required |
|---------------------------------------|----------------|----------|
| { | | |
| "replenishTargets": [{ | array (object) | |
| "cashunitTarget": Add example to YAML | string | |
| } | | |
| } | | |

Properties**replenishTargets**

Array of all suitable replenish targets. Empty if no suitable target was found.

replenishTargets/cashunitTarget

Object name of the cash unit (as stated by the [CashManagement.GetCashUnitInfo](#) command) that can be used as a target.

Event Messages

None

6.2.5 - CashAcceptor.GetDeviceLockStatus

This command is used to retrieve the lock/unlock statuses of the CashAcceptor device and each of its cash units. If the physical lock/unlock of both the CashAcceptor device and the cash units are not supported then an error will be returned.

Command Message

| Payload | Type | Required |
|--------------------------------------|------|----------|
| { "timeout": 5000 integer } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "deviceLockStatus": "lock", | string | |
| "cashUnitLock": [{ | array (object) | |
| "physicalPositionName": Add example to YAML, | string | |
| "cashUnitLockStatus": "lock" | string | |
| }] | | |

}

Properties

deviceLockStatus

Specifies the physical lock/unlock status of the CashAcceptor device. Following values are possible:

"lock": The device is physically locked.

"unlock": The device is physically unlocked.

"lockUnknown": Due to a hardware error or other condition, the physical lock/unlock status of the device cannot be determined.

"lockNotSupported": The Service does not support physical lock/unlock control of the device.

cashUnitLock

Array specifying the physical lock/unlock status of cash units. Cash units that do not support the physical lock/unlock control are not contained in the array. If there are no cash units that support physical lock/unlock control this will be empty.

cashUnitLock/physicalPositionName

A name identifying the physical location of the cash unit within the CashAcceptor. This name is the same as the "physicalPositionName" in the CashManagement.CashUnitInfo command.

cashUnitLock/cashUnitLockStatus

Specifies the physical lock/unlock status of cash units supported. Following values are possible:

"lock": The cash unit is physically locked.

"unlock": The cash unit is physically unlocked.

"lockUnknown": Due to a hardware error or other condition, the physical lock/unlock status of the cash unit cannot be determined.

Event Messages

None

6.2.6 - CashAcceptor.GetCashUnitCapabilities

This command is used to retrieve information on cash unit capabilities. It does not provide information on status or counters of cash units. This command can be seen as an extension to the CashManagement.CashUnitInfo command as it will always result in the same contents with regard to *number* and the cash unit information.

Command Message

| Payload | Type | Required |
|--------------------------------------|------|----------|
| { "timeout": 5000 integer } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "cashUnitCaps": { | object | |
| "additionalProperties": { | object | |
| "physicalPositionName": Add example to YAML, | string | |
| "maximumCapacity": 0, | integer | |
| "hardwareSensors": false, | boolean | |

```

"retractNoteCountThresholds": false,           boolean
"possibleItemTypes": {                         object
  "all": false,                                boolean
  "unfit": false,                               boolean
  "individual": false,                          boolean
  "level1": false,                             boolean
  "level2": false,                             boolean
  "level3": false,                             boolean
  "itemProcessor": false,                      boolean
  "unfitIndividual": false                   boolean
}
}
}
}
}

```

Properties

cashUnitCaps

Object containing additional cash unit capabilities. Cash Unit capability objects use the same names as used in [CashManagement.GetCashUnitInfo](#).

cashUnitCaps/additionalProperties

Cash unit capabilities.

cashUnitCaps/additionalProperties/physicalPositionName

A name identifying the physical location of the cash unit.

cashUnitCaps/additionalProperties/maximumCapacity

The maximum number of items the cash unit can hold. No threshold event

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

CashManagement.CashUnitThresholdEvent will be generated when this value is reached.
This value is persistent.

cashUnitCaps/additionalProperties/hardwareSensors

Specifies whether or not threshold events can be generated based on hardware sensors in the device. If this value is TRUE then threshold events may be generated based on hardware sensors as opposed to counts.

cashUnitCaps/additionalProperties/retractNoteCountThresholds

This field is only valid for cash units of type "retractCassette". It specifies whether the CashAcceptor retract cassette capacity is based on the number of notes, and therefore whether threshold events are generated based on note counts or the number of retract operations. If this value is set to TRUE, threshold events for retract cassettes are generated based on the number of notes, when *cashInCount* reaches the *maximum* value. If this value is set to FALSE, threshold events for retract cassettes are generated based on the number of retract operations, when *count* reaches the *maximum* value.

cashUnitCaps/additionalProperties/possibleItemTypes

Specifies the type of items the cash unit can be configured to accept as a combination of flags. The flags are defined as the same values listed in the *itemType* field of the CashManagement.CashUnitInfo output structure. The CashManagement.CashUnitInfo command describes the item types currently configured for a cash unit. This field provides the possible item types values that can be configured for a cash unit using the CashManagement.SetCashUnitInfo command.

cashUnitCaps/additionalProperties/possibleItemTypes/all

The cash unit takes all fit banknote types. These are level 4 notes which are fit for recycling.

cashUnitCaps/additionalProperties/possibleItemTypes/unfit

The cash unit takes all unfit banknotes. These are level 4 notes which are unfit for recycling.

cashUnitCaps/additionalProperties/possibleItemTypes/individual

The cash unit takes all types of fit banknotes specified in an individual list. These are level 4 notes which are fit for recycling.

`cashUnitCaps/additionalProperties/possibleItemTypes/level1`

Level 1 note types are stored in this cash unit.

`cashUnitCaps/additionalProperties/possibleItemTypes/level2`

If notes can be classified as level 2, then level 2 note types are stored in this cash unit.

`cashUnitCaps/additionalProperties/possibleItemTypes/level3`

If notes can be classified as level 3, then level 3 note types are stored in this cash unit.

`cashUnitCaps/additionalProperties/possibleItemTypes/itemProcessor`

The cash unit can accept items on the ItemProcessor interface.

`cashUnitCaps/additionalProperties/possibleItemTypes/unfitIndividual`

The cash unit takes all types of unfit banknotes specified in an individual list. These are level 4 notes which are unfit for recycling.

Event Messages

None

[6.2.7 - CashAcceptor.GetDepleteSource](#)

This command is used to determine which cash units can be specified as source cash units for a given target cash unit with the CashAcceptor.Deplete command. For example it can be used to determine which sources can be used for depletion to a replenishment container or to a cash-in cash unit.

Command Message

| Payload | Type | Required |
|---------|------|----------|
| { | | |

"cashunitTarget": Add example to YAML string

}

Properties

cashunitTarget

Object name of the cash unit (as stated by the [CashManagement.GetCashUnitInfo](#) command) which would be used as the target of the depletion operation.

Completion Message

| Payload | Type | Required |
|---|--------------------------|----------|
| { "depleteSources": [{ "cashunitSource": Add example to YAML }] } | array (object) string | |

Properties

depleteSources

Array of all suitable deplete sources. Empty if no suitable source was found.

depleteSources/cashunitSource

Object name of the cash unit (as stated by the [CashManagement.GetCashUnitInfo](#) command) that can be used as a source.

Event Messages

None

6.2.8 - CashAcceptor.GetCashUnitCountStatus

During normal processing it is possible that the *count* of a cash unit can become inaccurate due to a jam, mis-pick or other error situation. In this case the GetCashUnitCountStatus command could be used to report which cash units are known to have an inaccurate *count*. The client can then issue a CashAcceptor.CashUnitCount command for only those cash units if supported. Or alternatively the notes could be manually counted as part of a replenishment operation. This command returns the cash unit count status of all cash units.

Command Message

| Payload | Type | Required |
|--------------------------------------|------|----------|
| { "timeout": 5000 integer } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "cashUnitCountStatus": { | object | |
| "additionalProperties": { | object | |
| "physicalPositionName": Add example to YAML, | string | |
| "accuracy": "notSupported" | string | |
| } | | |
| } | | |

}

Properties

cashUnitCountStatus

Object containing cashUnitCountStatus objects. cashUnitCountStatus objects use the same names as used in [CashManagement.GetCashUnitInfo](#).

cashUnitCountStatus/additionalProperties/physicalPositionName

A name identifying the physical location of the cash unit within the CashAcceptor. This field can be used to identify shared cash units/media bins.

cashUnitCountStatus/additionalProperties/accuracy

Describes the accuracy of *count*. Following values are possible:

"notSupported": The hardware is not capable to determine the accuracy of *count*.

"accurate": The *count* is expected to be accurate. The notes were previously counted or replenished and there have since been no events that might have introduced inaccuracy. This value will be reported as a result of the following commands: Replenish and CashUnitCount.

"accurateSet": The *count* is expected to be accurate. The notes were previously set and there have since been no events that might have introduced inaccuracy.

"inaccurate": The *count* is likely to be inaccurate. A jam, picking fault, or some other event may have resulted in a counting inaccuracy.

"unknown": The accuracy of *count* cannot be determined. This may be due to cash unit insertion or some other hardware event.

Event Messages

None

6.2.9 - CashAcceptor.GetPresentStatus

This command is used to obtain the status of the most recent attempt to present or return items to the customer. This information includes the number of items previously moved to the output position and the number of items which have yet to be returned as a result of the following commands: CashIn, CashInRollback, PreparePresent, PresentMedia, OpenShutter (In the case of returning multiple bunches)

Command Message

| Payload | Type | Required |
|--------------------------------------|------|----------|
| { "timeout": 5000 integer } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|------------------------------|----------------|----------|
| { | | |
| "position": "left", | string | |
| "presentState": "presented", | string | |
| "additionalBunches": "none", | string | |
| "bunchesRemaining": 0, | integer | |
| "returnedItems": { | object | |
| "noteNumber": [{ | array (object) | |
| "noteID": 0, | integer | |

Note: work-in-progress. Use at your own risk.

"count": 0 integer

}

},

"totalReturnedItems": { object

See `returnedItems` properties.

},

"remainingItems": { object

See returnedItems properties.

}

}

Properties

position

Specifies the output position. Following values are possible:

"left": Left output position.

"right": Right output position.

"center": Center output position.

"top": Top output position.

"bottom": Bottom output position.

"front": Front output position.

"rear": Rear output position.

presentState

Supplies the status of the items that were to be presented by the most recent attempt to present or return items to the customer. Following values are possible:

"presented": The items were presented. This status is set as soon as the customer has access to the items.

"notPresented": The customer has not had access to the items.

"unknown": It is not known if the customer had access to the items.

additionalBunches

Specifies whether or not additional bunches of items are remaining to be presented as a result of the most recent operation. Following values are possible:

"none": No additional bunches remain.

"oneMore": At least one additional bunch remains.

"unknown": It is unknown whether additional bunches remain.

bunchesRemaining

If *additionalBunches* is "oneMore", specifies the number of additional bunches of items remaining to be presented as a result of the current operation. If the number of additional bunches is at least one, but the precise number is unknown, *bunchesRemaining* will be 255 (TODO: Check if there is a better way to represent this state). For any other value of *additionalBunches*, *bunchesRemaining* will be zero.

returnedItems

Array holding a list of banknote numbers which have been moved to the output position as a result of the most recent operation.

returnedItems/noteNumber

Array of banknote numbers the cash unit contains.

returnedItems/noteNumber/noteID

Identification of note type. The Note ID represents the note identifiers reported by the *CashAcceptor.BanknoteTypes* command. If this value is zero then the note type is unknown.

returnedItems/noteNumber/count

Actual count of cash items. The value is incremented each time cash items are moved to a cash unit. In the case of recycle cash units this count is decremented as defined in the description of the *logicalCount* field.

totalReturnedItems

Array of cumulative banknote numbers which have been moved to the output position. This value will be reset when the CashInStart, CashIn, CashInEnd, Retract, Reset or CashInRollback command is executed.

remainingItems

Array of banknote numbers on the intermediate stacker or transport which have not been yet moved to the output position.

Event Messages

None

6.2.10 - CashAcceptor.CashInStart

Before initiating a cash-in operation, a client must issue the CashAcceptor.CashInStart command to begin a cash-in transaction. During a cash-in transaction any number of CashAcceptor.CashIn commands may be issued. The transaction is ended when either a CashAcceptor.CashInRollback, CashAcceptor.CashInEnd, CashAcceptor.Retract or CashAcceptor.Reset command is sent. Where Capability *shutterControl* == FALSE this command precedes any explicit operation of the shutters.

CashAcceptor.Retract will terminate a transaction. In this case CashAcceptor.CashInEnd, CashAcceptor.CashInRollback and CashAcceptor.CashIn will report *noCashInActive*. If a client wishes to determine where the notes went during a transaction it can execute a CashUnitInfo before and after the transaction and then derive the difference.

A hardware failure during the cash-in transaction does not reset the note number list information; instead the note number list information will include items that could be accepted and identified up to the point of the hardware failure.

Exchange: This command can be used during an Exchange (*exchangeType* == depositInto) to deposit items accepted from the input position.

Command Message

| Payload | Type | Required |
|---------|------|----------|
| { | | |

```
"tellerID": 0,           integer  
"useRecycleUnits": false, boolean  
"outputPosition": "null", string  
"inputPosition": "null"   string  
}
```

Properties

tellerID

Identification of teller. This field is not applicable to Self-Service devices and should be omitted.

useRecycleUnits

Specifies whether or not the recycle cash units should be used when items are cashed in on a successful CashAcceptor.CashInEnd command. This parameter will be ignored if there are no recycle cash units or the hardware does not support this.

outputPosition

The output position where the items will be presented to the customer in the case of a rollback. Following values are possible:

"null": The items will be presented to the default configuration.

"left": The items will be presented to the left output position.

"right": The items will be presented to the right output position.

"center": The items will be presented to the center output position.

"top": The items will be presented to the top output position.

"bottom": The items will be presented to the bottom output position.

"front": The items will be presented to the front output position.

"rear": The items will be presented to the rear output position.

inputPosition

Specifies from which position the cash should be inserted. Following values are possible:

- "null": The cash is inserted from the default configuration.
- "left": The cash is inserted from the left input position.
- "right": The cash is inserted from the right input position.
- "center": The cash is inserted from the center input position.
- "top": The cash is inserted from the top input position.
- "bottom": The cash is inserted from the bottom input position.
- "front": The cash is inserted from the front input position.
- "rear": The cash is inserted from the rear input position.

Completion Message

| <u>Payload</u> | <u>Type</u> | <u>Required</u> |
|--|-------------|-----------------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "invalidTellerId" | string | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

"invalidTellerId": The teller ID is invalid. This error will never be generated by a Self-Service device.

"unsupportedPosition": The position specified is not supported.

"exchangeActive": The device is in the exchange state.

"cashInActive": The device is already in the cash-in state due to a previous CashAcceptor.CashInStart command.

"safeDoorOpen": The safe door is open. This device requires the safe door to be closed in order to perform a CashAcceptor.CashInStart command.

Event Messages

None

[6.2.11 - CashAcceptor.CashIn](#)

This command moves items into the cash device from an input position.

On devices with implicit shutter control, the CashAcceptor.InsertItems event will be generated when the device is ready to start accepting media.

The items may pass through the banknote reader for identification. Failure to identify items does not mean that the command has failed - even if some or all of the items are rejected by the banknote reader, the command may return *success*. In this case one or more CashAcceptor.InputRefuse events will be sent to report the rejection. See also paragraph below regarding returning refused items.

If the device does not have a banknote reader then the completion message will be empty.

If the device has a cash-in stacker then this command will cause inserted level 4 items to be moved there after validation. Level 2 and level 3 items may also be moved to the cash-in stacker, but some devices may immediately move them to a designated cash unit. Items on the stacker will remain there until the current cash-in transaction is either cancelled by the CashAcceptor.CashInRollback command or confirmed by the CashAcceptor.CashInEnd command. These commands will cause any level 2 or level 3 items on the cash-in stacker to be moved to the appropriate cash unit. If there is no cash-in stacker then this command will move items directly to the cash units and the CashAcceptor.CashInRollback command will not be supported. Cash unit information will be updated accordingly whenever notes are moved to a cash unit during this command.

Note that the *acceptor* status field may change value during a cash-in transaction. If media has been retained to cash units during a cash-in transaction, it may mean that *acceptor* is set to *stop*, which means subsequent cash-in operations may not be possible. In this case, the subsequent command fails with error code CashUnitError.

The *shutterControl* field of the capabilites structure returned from the Common.Capabilities query will determine whether the shutter is controlled implicitly by this command or whether the client must explicitly open and close the shutter using the OpenShutter and CloseShutter commands, or the CashAcceptor.PresentMedia command. If *shutterControl* is FALSE then this command does not operate the shutter in any way, the client is responsible for all shutter control. If *shutterControl* is TRUE this command opens the shutter at the start of the command and closes it once bills are inserted.

The *presentControl* field of the positionCapabilities structure returned from the CashAcceptor.PositionCapabilities query will determine whether or not it is necessary to call the CashAcceptor.PresentMedia command in order to move items to the output position. If *presentControl* is TRUE then all items are moved immediately to the correct output position for removal (a OpenShutter command will be needed in the case of explicit shutter control). If *presentControl* is FALSE then items are not returned immediately and must be presented to the correct output position for removal using the CashAcceptor.PresentMedia command.

It is possible that a device may divide bill or coin accepting into a series of sub-operations under hardware control. In this case a CashAcceptor.SubCashIn event may be sent after each sub-operation, if the hardware capabilities allow it.

Returning items (single bunch):

If *shutterControl* is TRUE, and a single bunch of items is returned then this command will complete once the notes have been returned. A CashAcceptor.ItemsPresented event will be generated.

If *shutterControl* is FALSE, and a single bunch of items is returned then this command will complete without generating a CashAcceptor.ItemsPresented event, instead the CashAcceptor.ItemsPresented event will be generated by the subsequent OpenShutter or CashAcceptor.PresentMedia command.

Returning items (multiple bunches):

It is possible that a device will in certain situations return refused items in multiple bunches. In this case, this command will not complete until the final bunch has been presented and after the last CashAcceptor.ItemsPresented event has been generated. For these devices *shutterControl* and *presentControl* fields of the Capabilities / positionCapabilities structure returned from the Common.Capabilities / CashAcceptor.PositionCapabilities query must both be TRUE otherwise it will not be possible to return multiple bunches. Additionally it may be possible to request the completion of this command with a Cancel before the final bunch is presented so that after the completion of this command the CashAcceptor.Retract or CashAcceptor.Reset

command can be used to move the remaining bunches, although the ability to do this will be hardware dependent.

Mixed Media Mode: If the device is operating in Mixed Media mode (Status *mixedMode* == mixedMedia) the Service Provider will not perform any operation unless the ItemProcessor.MediaIn command is called or has already been called on the ItemProcesspr interface.

Exchange: This command can be used during an Exchange (*exchangeType* == depositInto) to accept items from the input position.

Command Message

| Payload | Type | Required |
|-----------------|---------|----------|
| { | | |
| "timeout": 5000 | integer | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "cashUnitError", | string | |
| "noteNumber": [{ | array (object) | |
| "noteID": 0, | integer | |
| "count": 0 | integer | |

}]

}

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

"cashUnitError": A problem occurred with a cash unit. A CashManagement.CashUnitErrorEvent will be sent with the details.

"tooManyItems": There were too many items inserted previously. The cash-in stacker is full at the beginning of this command. This may also be reported where a limit specified by CashAcceptor.SetCashInLimit has already been reached at the beginning of this command.

"noItems": There were no items to cash-in.

"exchangeActive": The device is in an exchange state.

"shutterNotClosed": Shutter failed to close. In the case of explicit shutter control the client should close the shutter first.

"noCashInActive": There is no cash-in transaction active.

"positionNotempty": The output position is not empty so a cash-in is not possible.

"safeDoorOpen": The safe door is open. This device requires the safe door to be closed in order to perform a CashAcceptor.CashIn command.

"foreignItemsDetected": Foreign items have been detected inside the input position.

"shutterNotOpen": Shutter failed to open.

noteNumber

Array of banknote numbers the cash unit contains.

noteNumber/noteID

Identification of note type. The Note ID represents the note identifiers reported by the *CashAcceptor.BanknoteTypes* command. If this value is zero then the note type is unknown.

noteNumber/count

Actual count of cash items. The value is incremented each time cash items are moved to a cash unit. In the case of recycle cash units this count is decremented as defined in the description of the *logicalCount* field.

Event Messages

- [CashManagement.CashUnitErrorEvent](#)
- [CashAcceptor.InputRefuseEvent](#)
- [CashManagement.NoteErrorEvent](#)
- [CashAcceptor.SubCashInEvent](#)
- [CashAcceptor.ItemsInsertedEvent](#)
- [CashAcceptor.ItemsTakenEvent](#)
- [CashAcceptor.ItemsPresentedEvent](#)
- [CashManagement.InfoAvailableEvent](#)
- [CashAcceptor.InsertItemsEvent](#)
- [CashManagement.CashUnitThresholdEvent](#)
- [CashAcceptor.ShutterStatusChangedEvent](#)

[6.2.12 - CashAcceptor.CashInEnd](#)

This command ends a cash-in transaction. If cash items are on the stacker as a result of a *CashAcceptor.CashIn* command these items are moved to the appropriate cash units.

The cash-in transaction is ended even if this command does not complete successfully.

Mixed Media Mode:

If the device is operating in Mixed Media mode (Status *mixedMode == mixedMedia*) non-cash items, e.g. checks may be moved to an output position or media bin specified by the *ItemProcessor* interface. Additionally, the Service will not perform any operation unless the *ItemProcessor.MediaInEnd* command is called or has already been called on the

ItemProcessor. Alternatively, if Capabilities *mixedDepositAndRollback* is TRUE, then the ItemProcessor.MediaInRollback command could be used instead of the ItemProcessor.MediaInEnd command in order to deposit the bills and return the checks.

Where ItemProcessor items may be presented the *presentControl* field of the positionCapabilities structure returned from the CashAcceptor.PositionCapabilities query will determine whether or not it is necessary to call the CashAcceptor.PresentMedia command in order to move items to the output position. If *presentControl* is TRUE then all items are moved immediately to the correct output position for removal. If *presentControl* is FALSE then items are not returned immediately and must be presented to the correct output position for removal using the CashAcceptor.PresentMedia command.

Exchange: This command can be used during an Exchange (*exchangeType == depositInto*) to deposit items accepted from the input position.

Command Message

| Payload | Type | Required |
|--------------------------------------|------|----------|
| { "timeout": 5000 integer } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "cashUnitError", | string | |

| | |
|--|---------|
| "cashunits": { | object |
| "additionalProperties": { | object |
| "status": "ok", | string |
| "type": "billCassette", | string |
| "currencyID": Add example to YAML, | string |
| "value": 0, | number |
| "logicalCount": 0, | integer |
| "maximum": 0, | integer |
| "appLock": false, | boolean |
| "cashUnitName": Add example to YAML, | string |
| "initialCount": 0, | integer |
| "dispensedCount": 0, | integer |
| "presentedCount": 0, | integer |
| "retractedCount": 0, | integer |
| "rejectCount": 0, | integer |
| "minimum": 0, | integer |
| "physicalPositionName": Add example to YAML, | string |
| "unitID": Add example to YAML, | string |
| "count": 0, | integer |
| "maximumCapacity": 0, | integer |
| "hardwareSensor": false, | boolean |
| "itemType": { | object |
| "all": false, | boolean |
| "unfit": false, | boolean |
| "individual": false, | boolean |
| "level1": false, | boolean |

```

"level2": false,                                boolean
"level3": false,                                boolean
"itemProcessor": false,                           boolean
"unfitIndividual": false,                        boolean
},
"cashInCount": 0,                               integer
"noteNumberList": {                             object
    "noteNumber": [{                            array (object)
        "noteID": 0,                           integer
        "count": 0                           integer
    }]
},
"noteIDs": [0]                                   array (integer)
}
}
}

```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

"cashUnitError": A problem occurred with a cash unit. A CashManagement.CashUnitErrorEvent will be sent with the details.

"noItems": There were no items to cash-in.

"exchangeActive": The device is in an exchange state.

"noCashInActive": There is no cash-in transaction active.

"positionNotempty": The input or output position is not empty.

"safeDoorOpen": The safe door is open. This device requires the safe door to be closed in order to perform a CashAcceptor.CashInEnd command.

cashunits

Object containing cash unit information.

cashunits/additionalProperties

Cash unit information.

cashunits/additionalProperties/status

Supplies the status of the cash unit. Following values are possible:

- ok - The cash unit is in a good state.
- full - The cash unit is full.
- high - The cash unit is almost full (i.e. reached or exceeded the threshold defined by *maximum*).
- low - The cash unit is almost empty (i.e. reached or below the threshold defined by *minimum*).
- empty - The cash unit is empty, or insufficient items in the cash unit are preventing further dispense operations.
- inoperative - The cash unit is inoperative.
- missing - The cash unit is missing.
- noValue - The values of the specified cash unit are not available.
- noReference - There is no reference value available for the notes in this cash unit. The cash unit has not been calibrated.
- manipulated - The cash unit has been inserted (including removal followed by a reinsertion) when the device was not in the exchange state. This cash unit cannot be dispensed from.

cashunits/additionalProperties/type

Type of cash unit. Following values are possible:

- notApplicable - Not applicable. Typically means cash unit is missing.
- rejectCassette - Reject cash unit. This type will also indicate a combined reject/retract cash unit.
- billCassette - Cash unit containing bills.
- coinCylinder - Coin cylinder.
- coinDispenser - Coin dispenser as a whole unit.
- retractCassette - Retract cash unit.
- coupon - Cash unit containing coupons or advertising material.
- document - Cash unit containing documents.
- replenishmentContainer - Replenishment container. A cash unit can be refilled from a replenishment container.
- recycling - Recycling cash unit. This unit is only present when the device implements the Dispenser and CashAcceptor interfaces.
- cashIn - Cash-in cash unit.

cashunits/additionalProperties/currencyID

A three character string storing the ISO format [Ref. 2] Currency ID. This value will be omitted for cash units which contain items of more than one currency type or items to which currency is not applicable. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

cashunits/additionalProperties/value

Supplies the value of a single item in the cash unit. This value is expressed as floating point value. If the *currencyID* field for this cash unit is omitted, then this field will contain zero. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

cashunits/additionalProperties/logicalCount

The meaning of this count depends on the type of cash unit. This value is persistent. For all cash units except retract cash units (*type* is not *retractCassette*) this value specifies the number of items inside the cash unit. For all dispensing cash units (*type* is *billCassette*, *coinCylinder*, *coinDispenser*, *coupon*, *document* or *recycling*), this value includes any items from the cash unit not yet presented to the customer. This count is only decremented when the items are either known to be in customer access or successfully rejected. If the cash unit is usable from the CashAcceptor interface (*type* is *recycling*, *cashIn*, *retractCassette* or *rejectCassette*) then this value will be incremented as a result of a cash-in operation. Note

that for a reject cash unit (*type* is *rejectCassette*), this value is unreliable, since the typical reason for dumping items to the reject cash unit is a suspected count failure. For a retract cash unit (*type* is *retractCassette*) this value specifies the number of retract operations which result in items entering the cash unit.

cashunits/additionalProperties/maximum

When *count* reaches this value the threshold event

CashManagement.CashUnitThresholdEvent (*high*) will be generated. This value can be different from the actual capacity of the cassette. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

cashunits/additionalProperties/appLock

If this value is TRUE items cannot be dispensed from or deposited into the cash unit. If this value is TRUE and the client attempts to use the cash unit a CashManagement.CashUnitErrorEvent event will be generated and an error completion message will be returned. This value is persistent.

cashunits/additionalProperties/cashUnitName

A name which helps to identify the type of the cash unit. This is especially useful in the case of cash units of type *document* where different documents can have the same currency and value. For example, travelers checks and bank checks may have the same currency and value but still need to be identifiable as different types of document. Where this value is not relevant (e.g. in bill cash units) the property can be omitted. This value is persistent.

cashunits/additionalProperties/initialCount

Initial number of items contained in the cash unit. This value is persistent.

cashunits/additionalProperties/dispensedCount

The number of items dispensed from this cash unit. This count is incremented when the items are removed from the cash units. This count includes any items that were rejected during the dispense operation and are no longer in this cash unit. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

cashunits/additionalProperties/presentedCount

The number of items from this cash unit that have been presented to the customer. This count is incremented when the items are presented to the customer. If it is unknown if a customer has been presented with the items, then this count is not updated. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

cashunits/additionalProperties/retractedCount

The number of items that have been accessible to a customer and retracted into the cash unit. This value is persistent.

cashunits/additionalProperties/rejectCount

The number of items dispensed from this cash unit which have been rejected, are in a cash unit other than this cash unit, and which have not been accessible to a customer. This value may be unreliable, since a typical reason for rejecting items is a suspected pick failure. Other reasons for rejecting items may include incorrect note denominations, classifications not valid for dispensing, or where the transaction has been cancelled and a Reject command has been called. For reject and retract cash units (*type* is *rejectCassette* or *retractCassette*) this field does not apply and will be reported as zero. This value is persistent.

cashunits/additionalProperties/minimum

This field is not applicable to retract and reject cash units. For all cash units which dispense items (all other), when *count* reaches this value the threshold event CashManagement.CashUnitThresholdEvent (*low*) will be generated. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

cashunits/additionalProperties/physicalPositionName

A name identifying the physical location of the cash unit.

cashunits/additionalProperties/unitID

A 5 character string uniquely identifying the cash unit.

cashunits/additionalProperties/count

As defined by the *logicalCount* description, but with the following exceptions: This count does not include items dispensed but not yet presented. On cash units with *type* set to "retractCassette" the count represents the number of items, unless the device cannot count items during a retract, in which case this count will be zero. This value is persistent.

cashunits/additionalProperties/maximumCapacity

The maximum number of items the cash unit can hold. This is only for informational purposes. No threshold event CashManagement.CashUnitThresholdEvent will be generated. This value is persistent.

cashunits/additionalProperties/hardwareSensor

Specifies whether or not threshold events can be generated based on hardware sensors in the device. If this value is TRUE then threshold events may be generated based on hardware sensors as opposed to counts.

cashunits/additionalProperties/itemType

Specifies the type of items the cash unit takes as a combination of the following flags. The table in the Comments section of this command defines how to interpret the combination of these flags (TODO: include Table)

cashunits/additionalProperties/itemType/all

The cash unit takes all fit banknote types. These are level 4 notes which are fit for recycling.

cashunits/additionalProperties/itemType/unfit

The cash unit takes all unfit banknotes. These are level 4 notes which are unfit for recycling.

cashunits/additionalProperties/itemType/individual

The cash unit takes all types of fit banknotes specified in an individual list. These are level 4 notes which are fit for recycling.

cashunits/additionalProperties/itemType/level1

Level 1 note types are stored in this cash unit.

cashunits/additionalProperties/itemType/level2

If notes can be classified as level 2, then level 2 note types are stored in this cash unit.

cashunits/additionalProperties/itemType/level3

If notes can be classified as level 3, then level 3 note types are stored in this cash unit.

cashunits/additionalProperties/itemType/itemProcessor

The cash unit can accept items on the ItemProcessor interface.

cashunits/additionalProperties/itemType/unfitIndividual

The cash unit takes all types of unfit banknotes specified in an individual list. These are level 4 notes which are unfit for recycling.

cashunits/additionalProperties/cashInCount

Count of items that have entered the cash unit. This counter is incremented whenever an item enters a cash unit for any reason, unless it originated from this cash unit but was returned without being accessible to a customer. For a retract cash unit this value represents the total number of items of all types in the cash unit, or if the device cannot count items during a retract operation this value will be zero. This value is persistent.

cashunits/additionalProperties/noteNumberList

Array of cash items inside the cash unit. The content of this structure is persistent. If the cash unit is Dispenser specific cash unit with *type billCassette* or the contents of the cash unit are not known this structure will be omitted. If the cash unit is of *type retractCassette* this pointer will be omitted except for the following cases:

- If the retract cash unit is configured to accept level 2 notes then the number and type of level 2 notes is returned in the *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of level 2 notes.
- If items are recognized during retract operations then the number and type of notes retracted is returned in *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of retracted items.

`cashunits/additionalProperties/noteNumberList/noteNumber`

Array of banknote numbers the cash unit contains.

`cashunits/additionalProperties/noteNumberList/noteNumber/noteID`

Identification of note type. The Note ID represents the note identifiers reported by the *CashAcceptor.BanknoteTypes* command. If this value is zero then the note type is unknown.

`cashunits/additionalProperties/noteNumberList/noteNumber/count`

Actual count of cash items. The value is incremented each time cash items are moved to a cash unit. In the case of recycle cash units this count is decremented as defined in the description of the *logicalCount* field.

`cashunits/additionalProperties/noteIDs`

Array of integers which contains the note IDs of the banknotes the cash-in cash unit or recycle cash unit can take. This field only applies to *individual* cassette types. If there are no note IDs defined for the cassette or the cassette is not defined as *individual* then *noteIDs* will be omitted.

Event Messages

- [CashManagement.CashUnitThresholdEvent](#)
- [CashManagement.CashUnitInfoChangedEvent](#)
- [CashManagement.CashUnitErrorEvent](#)
- [CashManagement.InfoAvailableEvent](#)
- [CashManagement.NoteErrorEvent](#)
- [CashAcceptor.ItemsTakenEvent](#)
- [CashAcceptor.ItemsPresentedEvent](#)
- [CashAcceptor.ShutterStatusChangedEvent](#)

[6.2.13 - CashAcceptor.CashInRollback](#)

This command is used to roll back a cash-in transaction. It causes all the cash items cashed in since the last *CashAcceptor.CashInStart* command to be returned to the customer.

This command ends the current cash-in transaction. The cash-in transaction is ended even if this command does not complete successfully.

The *shutterControl* field of the Capabilities structure returned from the Common.Capabilities query will determine whether the shutter is controlled implicitly by this command or whether the client must explicitly control the shutter using the OpenShutter and CloseShutter commands, or CashAcceptor.PresentMedia command. If *shutterControl* is FALSE then this command does not operate the shutter in any way, the client is responsible for all shutter control. If *shutterControl* is TRUE then this command opens the shutter and it is closed when all items are removed.

The *presentControl* field of the positionCapabilities structure returned from the CashAcceptor.PositionCapabilities query will determine whether or not it is necessary to call the CashAcceptor.PresentMedia command in order to move items to the output position. If *presentControl* is TRUE then all items are moved immediately to the correct output position for removal (a OpenShutter command will be needed in the case of explicit shutter control). If *presentControl* is FALSE then items are not returned immediately and must be presented to the correct output position for removal using the CashAcceptor.PresentMedia command.

Items are returned in a single bunch or multiple bunches in the same way as described for the CashAcceptor.CashIn command.

Mixed Media Mode: If the device is operating in Mixed Media mode (Status *mixedMode* == mixedMedia) the Service will not perform any operation unless the ItemProcesspr.MediaInRollback command is called or has already been called on the ItemProcesspr interface. Alternatively, if the Capabilities *mixedDepositAndRollback* is TRUE, then the ItemProcessor.MediaInEnd command could be used instead of the ItemProcessor.MediaInRollback command in order to deposit the checks and return the items.

Exchange: This command can be used during an Exchange (*exchangeType* == depositInto) to return items accepted from the input position. Note that *ExchangeActive* would not be generated in this case.

Command Message

| Payload | Type | Required |
|-----------------|---------|----------|
| { | | |
| "timeout": 5000 | integer | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "cashUnitError", | string | |
| "cashunits": { | object | |
| "additionalProperties": { | object | |
| "status": "ok", | string | |
| "type": "billCassette", | string | |
| "currencyID": Add example to YAML, | string | |
| "value": 0, | number | |
| "logicalCount": 0, | integer | |
| "maximum": 0, | integer | |
| "appLock": false, | boolean | |
| "cashUnitName": Add example to YAML, | string | |
| "initialCount": 0, | integer | |
| "dispensedCount": 0, | integer | |
| "presentedCount": 0, | integer | |
| "retractedCount": 0, | integer | |
| "rejectCount": 0, | integer | |
| "minimum": 0, | integer | |
| "physicalPositionName": Add example to YAML, | string | |

```

"unitID": Add example to YAML,           string
"count": 0,                            integer
"maximumCapacity": 0,                  integer
"hardwareSensor": false,              boolean
"itemType": {                           object
  "all": false,                         boolean
  "unfit": false,                       boolean
  "individual": false,                  boolean
  "level1": false,                      boolean
  "level2": false,                      boolean
  "level3": false,                      boolean
  "itemProcessor": false,               boolean
  "unfitIndividual": false            boolean
},
"cashInCount": 0,                      integer
"noteNumberList": {                    object
  "noteNumber": [{                     array (object)
    "noteID": 0,                        integer
    "count": 0                          integer
  }]
},
"noteIDs": [0]                         array (integer)
}
}
}

```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

"cashUnitError": A problem occurred with a cash unit. A CashManagement.CashUnitErrorEvent will be sent with the details.

"shutterNotOpen": Shutter failed to open. In the case of explicit shutter control the client may have failed to open the shutter before issuing the command.

"exchangeActive": The device is in an exchange state.

"noCashInActive": There is no cash-in transaction active.

"positionNotEmpty": The input or output position is not empty.

"noItems": There were no items to rollback.

cashunits

Object containing cash unit information.

cashunits/additionalProperties

Cash unit information.

cashunits/additionalProperties/status

Supplies the status of the cash unit. Following values are possible:

- ok - The cash unit is in a good state.
- full - The cash unit is full.
- high - The cash unit is almost full (i.e. reached or exceeded the threshold defined by *maximum*).

- low - The cash unit is almost empty (i.e. reached or below the threshold defined by *minimum*).
- empty - The cash unit is empty, or insufficient items in the cash unit are preventing further dispense operations.
- inoperative - The cash unit is inoperative.
- missing - The cash unit is missing.
- noValue - The values of the specified cash unit are not available.
- noReference - There is no reference value available for the notes in this cash unit. The cash unit has not been calibrated.
- manipulated - The cash unit has been inserted (including removal followed by a reinsertion) when the device was not in the exchange state. This cash unit cannot be dispensed from.

cashunits/additionalProperties/type

Type of cash unit. Following values are possible:

- notApplicable - Not applicable. Typically means cash unit is missing.
- rejectCassette - Reject cash unit. This type will also indicate a combined reject/retract cash unit.
- billCassette - Cash unit containing bills.
- coinCylinder - Coin cylinder.
- coinDispenser - Coin dispenser as a whole unit.
- retractCassette - Retract cash unit.
- coupon - Cash unit containing coupons or advertising material.
- document - Cash unit containing documents.
- replenishmentContainer - Replenishment container. A cash unit can be refilled from a replenishment container.
- recycling - Recycling cash unit. This unit is only present when the device implements the Dispenser and CashAcceptor interfaces.
- cashIn - Cash-in cash unit.

cashunits/additionalProperties/currencyID

A three character string storing the ISO format [Ref. 2] Currency ID. This value will be omitted for cash units which contain items of more than one currency type or items to which currency is not applicable. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

cashunits/additionalProperties/value

Supplies the value of a single item in the cash unit. This value is expressed as floating point value. If the *currencyID* field for this cash unit is omitted, then this field will contain zero. If

the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

cashunits/additionalProperties/logicalCount

The meaning of this count depends on the type of cash unit. This value is persistent. For all cash units except retract cash units (*type* is not *retractCassette*) this value specifies the number of items inside the cash unit. For all dispensing cash units (*type* is *billCassette*, *coinCylinder*, *coinDispenser*, *coupon*, *document* or *recycling*), this value includes any items from the cash unit not yet presented to the customer. This count is only decremented when the items are either known to be in customer access or successfully rejected. If the cash unit is usable from the CashAcceptor interface (*type* is *recycling*, *cashIn*, *retractCassette* or *rejectCassette*) then this value will be incremented as a result of a cash-in operation. Note that for a reject cash unit (*type* is *rejectCassette*), this value is unreliable, since the typical reason for dumping items to the reject cash unit is a suspected count failure. For a retract cash unit (*type* is *retractCassette*) this value specifies the number of retract operations which result in items entering the cash unit.

cashunits/additionalProperties/maximum

When *count* reaches this value the threshold event

CashManagement.CashUnitThresholdEvent (*high*) will be generated. This value can be different from the actual capacity of the cassette. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

cashunits/additionalProperties/appLock

If this value is TRUE items cannot be dispensed from or deposited into the cash unit. If this value is TRUE and the client attempts to use the cash unit a CashManagement.CashUnitErrorEvent event will be generated and an error completion message will be returned. This value is persistent.

cashunits/additionalProperties/cashUnitName

A name which helps to identify the type of the cash unit. This is especially useful in the case of cash units of type *document* where different documents can have the same currency and value. For example, travelers checks and bank checks may have the same currency and value but still need to be identifiable as different types of document. Where this value is not relevant (e.g. in bill cash units) the property can be omitted. This value is persistent.

cashunits/additionalProperties/initialCount

Initial number of items contained in the cash unit. This value is persistent.

cashunits/additionalProperties/dispensedCount

The number of items dispensed from this cash unit. This count is incremented when the items are removed from the cash units. This count includes any items that were rejected during the dispense operation and are no longer in this cash unit. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

cashunits/additionalProperties/presentedCount

The number of items from this cash unit that have been presented to the customer. This count is incremented when the items are presented to the customer. If it is unknown if a customer has been presented with the items, then this count is not updated. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

cashunits/additionalProperties/retractedCount

The number of items that have been accessible to a customer and retracted into the cash unit. This value is persistent.

cashunits/additionalProperties/rejectCount

The number of items dispensed from this cash unit which have been rejected, are in a cash unit other than this cash unit, and which have not been accessible to a customer. This value may be unreliable, since a typical reason for rejecting items is a suspected pick failure. Other reasons for rejecting items may include incorrect note denominations, classifications not valid for dispensing, or where the transaction has been cancelled and a Reject command has been called. For reject and retract cash units (*type* is *rejectCassette* or *retractCassette*) this field does not apply and will be reported as zero. This value is persistent.

cashunits/additionalProperties/minimum

This field is not applicable to retract and reject cash units. For all cash units which dispense items (all other), when *count* reaches this value the threshold event CashManagement.CashUnitThresholdEvent (*low*) will be generated. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value

is persistent.

cashunits/additionalProperties/physicalPositionName

A name identifying the physical location of the cash unit.

cashunits/additionalProperties/unitID

A 5 character string uniquely identifying the cash unit.

cashunits/additionalProperties/count

As defined by the *logicalCount* description, but with the following exceptions: This count does not include items dispensed but not yet presented. On cash units with *type* set to "retractCassette" the count represents the number of items, unless the device cannot count items during a retract, in which case this count will be zero. This value is persistent.

cashunits/additionalProperties/maximumCapacity

The maximum number of items the cash unit can hold. This is only for informational purposes. No threshold event CashManagement.CashUnitThresholdEvent will be generated. This value is persistent.

cashunits/additionalProperties/hardwareSensor

Specifies whether or not threshold events can be generated based on hardware sensors in the device. If this value is TRUE then threshold events may be generated based on hardware sensors as opposed to counts.

cashunits/additionalProperties/itemType

Specifies the type of items the cash unit takes as a combination of the following flags. The table in the Comments section of this command defines how to interpret the combination of these flags (TODO: include Table)

cashunits/additionalProperties/itemType/all

The cash unit takes all fit banknote types. These are level 4 notes which are fit for recycling.

`cashunits/additionalProperties/itemType/unfit`

The cash unit takes all unfit banknotes. These are level 4 notes which are unfit for recycling.

`cashunits/additionalProperties/itemType/individual`

The cash unit takes all types of fit banknotes specified in an individual list. These are level 4 notes which are fit for recycling.

`cashunits/additionalProperties/itemType/level1`

Level 1 note types are stored in this cash unit.

`cashunits/additionalProperties/itemType/level2`

If notes can be classified as level 2, then level 2 note types are stored in this cash unit.

`cashunits/additionalProperties/itemType/level3`

If notes can be classified as level 3, then level 3 note types are stored in this cash unit.

`cashunits/additionalProperties/itemType/itemProcessor`

The cash unit can accept items on the ItemProcessor interface.

`cashunits/additionalProperties/itemType/unfitIndividual`

The cash unit takes all types of unfit banknotes specified in an individual list. These are level 4 notes which are unfit for recycling.

`cashunits/additionalProperties/cashInCount`

Count of items that have entered the cash unit. This counter is incremented whenever an item enters a cash unit for any reason, unless it originated from this cash unit but was returned without being accessible to a customer. For a retract cash unit this value represents the total number of items of all types in the cash unit, or if the device cannot count items during a retract operation this value will be zero. This value is persistent.

cashunits/additionalProperties/noteNumberList

Array of cash items inside the cash unit. The content of this structure is persistent. If the cash unit is Dispenser specific cash unit with *type billCassette* or the contents of the cash unit are not known this structure will be omitted. If the cash unit is of *type retractCassette* this pointer will be omitted except for the following cases:

- If the retract cash unit is configured to accept level 2 notes then the number and type of level 2 notes is returned in the *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of level 2 notes.
- If items are recognized during retract operations then the number and type of notes retracted is returned in *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of retracted items.

cashunits/additionalProperties/noteNumberList/noteNumber

Array of banknote numbers the cash unit contains.

cashunits/additionalProperties/noteNumberList/noteNumber/noteID

Identification of note type. The Note ID represents the note identifiers reported by the *CashAcceptor.BanknoteTypes* command. If this value is zero then the note type is unknown.

cashunits/additionalProperties/noteNumberList/noteNumber/count

Actual count of cash items. The value is incremented each time cash items are moved to a cash unit. In the case of recycle cash units this count is decremented as defined in the description of the *logicalCount* field.

cashunits/additionalProperties/noteIDs

Array of integers which contains the note IDs of the banknotes the cash-in cash unit or recycle cash unit can take. This field only applies to *individual* cassette types. If there are no note IDs defined for the cassette or the cassette is not defined as *individual* then *noteIDs* will be omitted.

Event Messages

- [CashManagement.CashUnitErrorEvent](#)
- [CashAcceptor.ItemsTakenEvent](#)
- [CashAcceptor.ItemsPresentedEvent](#)

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

- [CashManagement.InfoAvailableEvent](#)
- [CashManagement.CashUnitThresholdEvent](#)
- [CashAcceptor.ShutterStatusChangedEvent](#)

6.2.14 - [CashAcceptor.Retract](#)

This command retracts items from an output position or internal areas within the device. Retracted items will be moved to either a retract bin, a reject bin, cash-in/recycle cash units, the transport or an intermediate stacker area. If items from internal areas within the device are preventing items at an output position from being retracted then the items from the internal areas will be retracted first. When the items are retracted from an output position the shutter is closed automatically, even if the *shutterControl* capability is set to FALSE.

This command terminates a running cash-in transaction. The cash-in transaction is terminated even if this command does not complete successfully.

Mixed Media Mode:

If the device is operating in Mixed Media mode (Status *mixedMode* == mixedMedia) this command will not perform any operation unless the ItemProcessor.RetractMedia command is called or has already been called on the ItemProcessor interface. Where the parameters for this command and the corresponding ItemProcessor.RetractMedia command conflict, for example the device is physically unable to satisfy both commands, the CashAcceptor.Retract input parameters will be used for all items.

Exchange: This command can be used during an Exchange (*exchangeType* == depositInto) to retract items. Note that an ExchangeActive error would not be generated in this case.

Command Message

| Payload | Type | Required |
|---------------------------|---------|----------|
| { | | |
| "outputPosition": "null", | string | |
| "retractArea": "retract", | string | |
| "index": 0 | integer | |
| } | | |

Properties

outputPosition

Specifies the output position from which to retract the bills. Following values are possible:

"null": The default configuration information should be used. This value is also used to retract items from internal device locations.

"left": Retract items from the left output position.

"right": Retract items from the right output position.

"center": Retract items from the center output position.

"top": Retract items from the top output position.

"bottom": Retract items from the bottom output position.

"front": Retract items from the front output position.

"rear": Retract items from the rear output position.

retractArea

This value specifies the area to which the items are to be retracted. Following values are possible:

"retract": Retract the items to a retract cash unit.

"reject": Retract the items to a reject cash unit.

"transport": Retract the items to the transport.

"stacker": Retract the items to the intermediate stacker area.

"billCassettes": Retract the items to item cassettes, i.e. cash-in and recycle cash units.

"cashIn": Retract the items to a cash-in cash unit. The *itemType* of the cash-in cash unit defined in CashManagement.CashUnitInfo must include "all" and "unfit".

index

If *retractArea* is set to "retract" this field defines the position inside the retract cash units into which the cash is to be retracted. *index* starts with a value of one (1) for the first retract position and increments by one for each subsequent position. The maximum value of *index* is the sum of the *maximum* of each retract cash unit.

If *retractArea* is set to "cashIn" this field defines the cash unit under the "cashIn" cash units into which the cash is to be retracted. *index* corresponds to the cash unit *number* defined in

`CashManagement.CashUnitInfo`.

If `retractArea` is not set to "retract" or "cashIn" then the value of this field is ignored.

Completion Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "cashUnitError", | string | |
| "cashunits": { | object | |
| "additionalProperties": { | object | |
| "status": "ok", | string | |
| "type": "billCassette", | string | |
| "currencyID": Add example to YAML, | string | |
| "value": 0, | number | |
| "logicalCount": 0, | integer | |
| "maximum": 0, | integer | |
| "appLock": false, | boolean | |
| "cashUnitName": Add example to YAML, | string | |
| "initialCount": 0, | integer | |
| "dispensedCount": 0, | integer | |
| "presentedCount": 0, | integer | |
| "retractedCount": 0, | integer | |
| "rejectCount": 0, | integer | |
| "minimum": 0, | integer | |
| "physicalPositionName": Add example to YAML, | string | |
| "unitID": Add example to YAML, | string | |

```

"count": 0,                                integer
"maximumCapacity": 0,                      integer
"hardwareSensor": false,                   boolean
"itemType": {                               object
    "all": false,                           boolean
    "unfit": false,                        boolean
    "individual": false,                   boolean
    "level1": false,                       boolean
    "level2": false,                       boolean
    "level3": false,                       boolean
    "itemProcessor": false,                boolean
    "unfitIndividual": false             boolean
},
"cashInCount": 0,                            integer
"noteNumberList": {                         object
    "noteNumber": [{                        array (object)
        "noteID": 0,                          integer
        "count": 0                           integer
    }]
},
"noteIDs": [0]                                array (integer)
}
}
}

```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

"cashUnitError": A problem occurred with a cash unit. A CashManagement.CashUnitErrorEvent will be sent with the details.

"noItems": There were no items to retract.

"exchangeActive": The device is in an exchange state.

"shutterNotClosed": The shutter failed to close.

"itemsTaken": Items were present at the output position at the start of the operation, but were removed before the operation was complete - some or all of the items were not retracted.

"invalidRetractPosition": The *index* is not supported.

"notRetractArea": The retract area specified in *retractArea* is not supported.

"foreignItemsDetected": Foreign items have been detected inside the input position.

cashunits

Object containing cash unit information.

cashunits/additionalProperties

Cash unit information.

cashunits/additionalProperties/status

Supplies the status of the cash unit. Following values are possible:

- ok - The cash unit is in a good state.

- full - The cash unit is full.
- high - The cash unit is almost full (i.e. reached or exceeded the threshold defined by *maximum*).
- low - The cash unit is almost empty (i.e. reached or below the threshold defined by *minimum*).
- empty - The cash unit is empty, or insufficient items in the cash unit are preventing further dispense operations.
- inoperative - The cash unit is inoperative.
- missing - The cash unit is missing.
- noValue - The values of the specified cash unit are not available.
- noReference - There is no reference value available for the notes in this cash unit. The cash unit has not been calibrated.
- manipulated - The cash unit has been inserted (including removal followed by a reinsertion) when the device was not in the exchange state. This cash unit cannot be dispensed from.

cashunits/additionalProperties/type

Type of cash unit. Following values are possible:

- notApplicable - Not applicable. Typically means cash unit is missing.
- rejectCassette - Reject cash unit. This type will also indicate a combined reject/retract cash unit.
- billCassette - Cash unit containing bills.
- coinCylinder - Coin cylinder.
- coinDispenser - Coin dispenser as a whole unit.
- retractCassette - Retract cash unit.
- coupon - Cash unit containing coupons or advertising material.
- document - Cash unit containing documents.
- replenishmentContainer - Replenishment container. A cash unit can be refilled from a replenishment container.
- recycling - Recycling cash unit. This unit is only present when the device implements the Dispenser and CashAcceptor interfaces.
- cashIn - Cash-in cash unit.

cashunits/additionalProperties/currencyID

A three character string storing the ISO format [Ref. 2] Currency ID. This value will be omitted for cash units which contain items of more than one currency type or items to which currency is not applicable. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

cashunits/additionalProperties/value

Supplies the value of a single item in the cash unit. This value is expressed as floating point value. If the *currencyID* field for this cash unit is omitted, then this field will contain zero. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

cashunits/additionalProperties/logicalCount

The meaning of this count depends on the type of cash unit. This value is persistent. For all cash units except retract cash units (*type* is not *retractCassette*) this value specifies the number of items inside the cash unit. For all dispensing cash units (*type* is *billCassette*, *coinCylinder*, *coinDispenser*, *coupon*, *document* or *recycling*), this value includes any items from the cash unit not yet presented to the customer. This count is only decremented when the items are either known to be in customer access or successfully rejected. If the cash unit is usable from the CashAcceptor interface (*type* is *recycling*, *cashIn*, *retractCassette* or *rejectCassette*) then this value will be incremented as a result of a cash-in operation. Note that for a reject cash unit (*type* is *rejectCassette*), this value is unreliable, since the typical reason for dumping items to the reject cash unit is a suspected count failure. For a retract cash unit (*type* is *retractCassette*) this value specifies the number of retract operations which result in items entering the cash unit.

cashunits/additionalProperties/maximum

When *count* reaches this value the threshold event

`CashManagement.CashUnitThresholdEvent (high)` will be generated. This value can be different from the actual capacity of the cassette. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

cashunits/additionalProperties/appLock

If this value is TRUE items cannot be dispensed from or deposited into the cash unit. If this value is TRUE and the client attempts to use the cash unit a `CashManagement.CashUnitErrorEvent` event will be generated and an error completion message will be returned. This value is persistent.

cashunits/additionalProperties/cashUnitName

A name which helps to identify the type of the cash unit. This is especially useful in the case of cash units of type *document* where different documents can have the same currency and value. For example, travelers checks and bank checks may have the same currency and value but still need to be identifiable as different types of document. Where this value is not

relevant (e.g. in bill cash units) the property can be omitted. This value is persistent.

cashunits/additionalProperties/initialCount

Initial number of items contained in the cash unit. This value is persistent.

cashunits/additionalProperties/dispensedCount

The number of items dispensed from this cash unit. This count is incremented when the items are removed from the cash units. This count includes any items that were rejected during the dispense operation and are no longer in this cash unit. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

cashunits/additionalProperties/presentedCount

The number of items from this cash unit that have been presented to the customer. This count is incremented when the items are presented to the customer. If it is unknown if a customer has been presented with the items, then this count is not updated. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

cashunits/additionalProperties/retractedCount

The number of items that have been accessible to a customer and retracted into the cash unit. This value is persistent.

cashunits/additionalProperties/rejectCount

The number of items dispensed from this cash unit which have been rejected, are in a cash unit other than this cash unit, and which have not been accessible to a customer. This value may be unreliable, since a typical reason for rejecting items is a suspected pick failure. Other reasons for rejecting items may include incorrect note denominations, classifications not valid for dispensing, or where the transaction has been cancelled and a Reject command has been called. For reject and retract cash units (*type* is *rejectCassette* or *retractCassette*) this field does not apply and will be reported as zero. This value is persistent.

cashunits/additionalProperties/minimum

This field is not applicable to retract and reject cash units. For all cash units which dispense items (all other), when *count* reaches this value the threshold event

CashManagement.CashUnitThresholdEvent (*low*) will be generated. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

cashunits/additionalProperties/physicalPositionName

A name identifying the physical location of the cash unit.

cashunits/additionalProperties/unitID

A 5 character string uniquely identifying the cash unit.

cashunits/additionalProperties/count

As defined by the *logicalCount* description, but with the following exceptions: This count does not include items dispensed but not yet presented. On cash units with *type* set to "retractCassette" the count represents the number of items, unless the device cannot count items during a retract, in which case this count will be zero. This value is persistent.

cashunits/additionalProperties/maximumCapacity

The maximum number of items the cash unit can hold. This is only for informational purposes. No threshold event CashManagement.CashUnitThresholdEvent will be generated. This value is persistent.

cashunits/additionalProperties/hardwareSensor

Specifies whether or not threshold events can be generated based on hardware sensors in the device. If this value is TRUE then threshold events may be generated based on hardware sensors as opposed to counts.

cashunits/additionalProperties/itemType

Specifies the type of items the cash unit takes as a combination of the following flags. The table in the Comments section of this command defines how to interpret the combination of these flags (TODO: include Table)

cashunits/additionalProperties/itemType/all

The cash unit takes all fit banknote types. These are level 4 notes which are fit for recycling.

cashunits/additionalProperties/itemType/unfit

The cash unit takes all unfit banknotes. These are level 4 notes which are unfit for recycling.

cashunits/additionalProperties/itemType/individual

The cash unit takes all types of fit banknotes specified in an individual list. These are level 4 notes which are fit for recycling.

cashunits/additionalProperties/itemType/level1

Level 1 note types are stored in this cash unit.

cashunits/additionalProperties/itemType/level2

If notes can be classified as level 2, then level 2 note types are stored in this cash unit.

cashunits/additionalProperties/itemType/level3

If notes can be classified as level 3, then level 3 note types are stored in this cash unit.

cashunits/additionalProperties/itemType/itemProcessor

The cash unit can accept items on the ItemProcessor interface.

cashunits/additionalProperties/itemType/unfitIndividual

The cash unit takes all types of unfit banknotes specified in an individual list. These are level 4 notes which are unfit for recycling.

cashunits/additionalProperties/cashInCount

Count of items that have entered the cash unit. This counter is incremented whenever an item enters a cash unit for any reason, unless it originated from this cash unit but was returned without being accessible to a customer. For a retract cash unit this value represents the total number of items of all types in the cash unit, or if the device cannot

count items during a retract operation this value will be zero. This value is persistent.

cashunits/additionalProperties/noteNumberList

Array of cash items inside the cash unit. The content of this structure is persistent. If the cash unit is Dispenser specific cash unit with *type billCassette* or the contents of the cash unit are not known this structure will be omitted. If the cash unit is of *type retractCassette* this pointer will be omitted except for the following cases:

- If the retract cash unit is configured to accept level 2 notes then the number and type of level 2 notes is returned in the *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of level 2 notes.
- If items are recognized during retract operations then the number and type of notes retracted is returned in *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of retracted items.

cashunits/additionalProperties/noteNumberList/noteNumber

Array of banknote numbers the cash unit contains.

cashunits/additionalProperties/noteNumberList/noteNumber/noteID

Identification of note type. The Note ID represents the note identifiers reported by the *CashAcceptor.BanknoteTypes* command. If this value is zero then the note type is unknown.

cashunits/additionalProperties/noteNumberList/noteNumber/count

Actual count of cash items. The value is incremented each time cash items are moved to a cash unit. In the case of recycle cash units this count is decremented as defined in the description of the *logicalCount* field.

cashunits/additionalProperties/noteIDs

Array of integers which contains the note IDs of the banknotes the cash-in cash unit or recycle cash unit can take. This field only applies to *individual* cassette types. If there are no note IDs defined for the cassette or the cassette is not defined as *individual* then *noteIDs* will be omitted.

Event Messages

- [CashManagement.CashUnitThresholdEvent](#)

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

- [CashManagement.CashUnitErrorEvent](#)
- [CashManagement.NoteErrorEvent](#)
- [CashAcceptor.ItemsTakenEvent](#)
- [CashManagement.InfoAvailableEvent](#)
- [CashManagement.CashUnitInfoChangedEvent](#)
- [CashAcceptor.ShutterStatusChangedEvent](#)

6.2.15 - CashAcceptor.OpenShutter

This command opens the shutter. In cases where multiple bunches are to be returned under explicit shutter control and the first bunch has already been presented and taken and the output position is empty, this command moves the next bunch to the output position before opening the shutter -- see sections 8.6 and 8.7. This does not apply if the output position is not empty, for example if items had been re-inserted or dropped back into the output position as the shutter closed.

Command Message

| Payload | Type | Required |
|--------------------------------------|------|----------|
| { "position": "null" string } | | |

Properties

position

Position where the shutter is to be opened. If the client does not need to specify the shutter, this field can be omitted or set to "null". Otherwise this field should be set to one of the following values:

"null": The default configuration information should be used.

"inLeft": Open the shutter of the left input position.

"inRight": Open the shutter of the right input position.

"inCenter": Open the shutter of the center input position.

"inTop": Open the shutter of the top input position.

"inBottom": Open the shutter of the bottom input position.

"inFront": Open the shutter of the front input position.

"inRear": Open the shutter of the rear input position.

"outLeft": Open the shutter of the left output position.

"outRight": Open the shutter of the right output position.

"outCenter": Open the shutter of the center output position.

"outTop": Open the shutter of the top output position.

"outBottom": Open the shutter of the bottom output position.

"outFront": Open the shutter of the front output position.

"outRear": Open the shutter of the rear output position.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "unsupportedPosition" | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

"unsupportedPosition": The position specified is not supported.

"shutterNotOpen": Shutter failed to open.

"shutterOpen": Shutter was already open.

"exchangeActive": The device is in an exchange state. Note that this would not apply during an Exchange (*exchangeType == "depositInto"*).

"foreignItemsDetected": Foreign items have been detected in the input position.

Event Messages

- [CashAcceptor.ItemsTakenEvent](#)
- [CashAcceptor.ItemsInsertedEvent](#)
- [CashAcceptor.ShutterStatusChangedEvent](#)

[6.2.16 - CashAcceptor.CloseShutter](#)

This command closes the shutter.

Command Message

| Payload | Type | Required |
|--------------------------------------|------|----------|
| { "position": "null" string } | | |

Properties

position

Position where the shutter is to be closed. If the client does not need to specify the shutter, this field can be omitted or set to "null". Otherwise this field should be set to one of the following values:

"null": The default configuration information should be used.

"inLeft": Close the shutter of the left input position.

"inRight": Close the shutter of the right input position.

"inCenter": Close the shutter of the center input position.

"inTop": Close the shutter of the top input position.

"inBottom": Close the shutter of the bottom input position.

"inFront": Close the shutter of the front input position.

"inRear": Close the shutter of the rear input position.

"outLeft": Close the shutter of the left output position.

"outRight": Close the shutter of the right output position.

"outCenter": Close the shutter of the center output position.

"outTop": Close the shutter of the top output position.

"outBottom": Close the shutter of the bottom output position.

"outFront": Close the shutter of the front output position.

"outRear": Close the shutter of the rear output position.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "unsupportedPosition" | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

"unsupportedPosition": The position specified is not supported.

"shutterClosed": Shutter was already closed.

"exchangeActive": The device is in an exchange state. Note that this would not apply during an Exchange (*exchangeType == "depositInto"*).

"shutterNotClosed": Shutter failed to close.

"tooManyItems": There were too many items inserted for the shutter to close.

"foreignItemsDetected": Foreign items have been detected in the input position. The shutter is open.

Event Messages

- [CashAcceptor.ShutterStatusChangedEvent](#)

[6.2.17 - CashAcceptor.Reset](#)

This command is used by the client to perform a hardware reset which will attempt to return the device to a known good state. This command does not over-ride a lock obtained on another client or service handle.

If a cash-in transaction is active, this command will end it (even if this command does not complete successfully). If an exchange state is active then this command will end the exchange state (even if this command does not complete successfully).

Persistent values, such as counts and configuration information are not cleared by this command.

The device will attempt to move any items found anywhere within the device to the position specified within the *resetIn* parameter. This may not always be possible because of hardware problems.

If items are found inside the device one or more CashAcceptor.MediaDetected events will be generated to inform the client where the items have actually been moved to.

The *shutterControl* field of the Capabilities structure returned from the Common.Capabilities query will determine whether the shutter is controlled implicitly by this command or whether the client must explicitly control the shutter using the OpenShutter and CloseShutter commands, or the CashAcceptor.PresentMedia command. If *shutterControl* is FALSE then this command does not operate the shutter in any way, the client is responsible for all shutter control. If *shutterControl* is TRUE then this command operates the shutter as necessary so that the shutter is closed after the command completes successfully and any items returned to the customer have been removed.

The *presentControl* field of the *positionCapabilities* structure returned from the CashAcceptor.PositionCapabilities query will determine whether or not it is necessary to call the CashAcceptor.PresentMedia command in order to move items to the output position. If *presentControl* is TRUE then all items are moved immediately to the correct output position for removal (a OpenShutter command will be needed in the case of explicit shutter control). If *presentControl* is FALSE then items are not returned immediately and must be presented to the correct output position for removal using the CashAcceptor.PresentMedia command.

If requested, items are returned in a single bunch or multiple bunches in the same way as described for the CashAcceptor.CashIn command.

Mixed Media Mode:

The value of Status *mixedMode* is not changed by this command. Where the items are to be moved to a cash unit, the cash unit must support an *itemType* of "itemProcessor".

Command Message

| Payload | Type | Required |
|----------------------------------|---------|----------|
| { | | |
| "cashunit": Add example to YAML, | string | |
| "retractArea": { | object | |
| "outputPosition": "null", | string | |
| "retractArea": "retract", | string | |
| "index": 0 | integer | |
| }, | | |
| "outputPosition": "null" | string | |
| } | | |

Properties

cashunit

If defined, this value specifies the object name (as stated by the [CashManagement.GetCashUnitInfo](#) command) of the single cash unit to be used for the storage of any items found.

If items are to be moved to an output position, this value must be omitted, *retractArea* must be omitted and *outputPosition* specifies where items are to be moved to. If this value is omitted and items are to be moved to internal areas of the device, *retractArea* specifies where items are to be moved to or stored.

retractArea

This field is used if items are to be moved to internal areas of the device, including cash units, the intermediate stacker or the transport. The field is only relevant if *cashunit* is not defined.

retractArea/outputPosition

Specifies the output position from which to retract the bills. Following values are possible:

"null": The default configuration information should be used. This value is also used to retract items from internal device locations.

"left": Retract items from the left output position.

"right": Retract items from the right output position.

"center": Retract items from the center output position.

"top": Retract items from the top output position.

"bottom": Retract items from the bottom output position.

"front": Retract items from the front output position.

"rear": Retract items from the rear output position.

retractArea/retractArea

This value specifies the area to which the items are to be retracted. Following values are possible:

"retract": Retract the items to a retract cash unit.

"reject": Retract the items to a reject cash unit.

"transport": Retract the items to the transport.

"stacker": Retract the items to the intermediate stacker area.

"billCassettes": Retract the items to item cassettes, i.e. cash-in and recycle cash units.

"cashIn": Retract the items to a cash-in cash unit. The *itemType* of the cash-in cash unit defined in CashManagement.CashUnitInfo must include "all" and "unfit".

retractArea/index

If *retractArea* is set to "retract" this field defines the position inside the retract cash units into which the cash is to be retracted. *index* starts with a value of one (1) for the first retract position and increments by one for each subsequent position. The maximum value of *index* is the sum of the *maximum* of each retract cash unit.

If *retractArea* is set to "cashIn" this field defines the cash unit under the "cashIn" cash units into which the cash is to be retracted. *index* corresponds to the cash unit *number* defined in CashManagement.CashUnitInfo.

If *retractArea* is not set to "retract" or "cashIn" then the value of this field is ignored.

outputPosition

The output position to which items are to be moved. This field is only used if *number* is zero and *retractArea* is omitted. Following values are possible:

"null": Take the default configuration.

"left": Move items to the left output position.

"right": Move items to the right output position.

"center": Move items to the center output position.

"top": Move items to the top output position.

"bottom": Move items to the bottom output position.

"front": Move items to the front output position.

"rear": Move items to the rear output position.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "cashUnitError" | string | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

"cashUnitError": A cash unit caused an error. A CashManagement.CashUnitErrorEvent will be sent with the details.

"unsupportedPosition": The position specified is not supported.

"invalidCashUnit": The cash unit number specified is not valid.

"invalidRetractPosition": The *index* is not supported.

"notRetractArea": The retract area specified in *retractArea* is not supported.

"foreignItemsDetected": Foreign items have been detected in the input position.

Event Messages

- [CashManagement.CashUnitThresholdEvent](#)
- [CashManagement.CashUnitErrorEvent](#)

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

- [CashAcceptor.MediaDetectedEvent](#)
- [CashAcceptor.ItemsTakenEvent](#)
- [CashManagement.InfoAvailableEvent](#)
- [CashAcceptor.ShutterStatusChangedEvent](#)

[6.2.18 - CashAcceptor.ConfigureNotetypes](#)

This command is used to configure the note types the banknote reader should accept during cash-in. All note types the banknote reader should accept must be given in the input structure. If an unknown note type is given the error code "unsupportedData" will be returned. The values set by this command are persistent.

Command Message

| Payload | Type | Required |
|---|------|----------|
| { "noteIDs": [0] array (integer) } | | |

Properties

noteIDs

Array of unsigned integers which contains the note IDs of the banknotes the banknote reader can accept.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "exchangeActive" | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

"exchangeActive": The device is in the exchange state.

"cashInActive": A cash-in transaction is active. This device requires that no cash-in transaction is active in order to perform the command.

Event Messages

None

[6.2.19 - CashAcceptor.CreateP6Signature](#)

This command is used to create a reference signature (normally a level 3 note) that was checked and regarded as a forgery. The reference can be compared with the available signatures of the cash-in transactions to track back the customer.

When this command is executed, the device waits for a note to be inserted at the input position, transports the note to the recognition module, creates the signature and then returns the note to the output position.

The *shutterControl* field of the *capabilities* structure returned from the Common.Capabilities query will determine whether the shutter is controlled implicitly by this command or whether the client must explicitly control the shutter using the OpenShutter and CloseShutter commands, or CashAcceptor.PresentMedia command. If *shutterControl* is FALSE then this command does not operate the shutter in any way, the client is responsible for all shutter control. If *shutterControl* is TRUE then this command opens and closes the shutter at various times during the command execution and the shutter is finally closed when all items are removed.

The *presentControl* field of the *positionCapabilities* structure returned from the CashAcceptor.PositionCapabilities query will determine whether or not it is necessary to call the CashAcceptor.PresentMedia command in order to move items to the output position. If *presentControl* is TRUE then all items are moved immediately to the correct output position for removal (a OpenShutter command will be needed in the case of explicit shutter control). If *presentControl* is FALSE then items are not returned immediately and must be presented to the correct output position for removal using the CashAcceptor.PresentMedia command.

On devices with implicit shutter control, the InsertItems event will be generated when the device is ready to start accepting media.

The client may have to execute this command repeatedly to make sure that all possible signatures are captured.

If a single note is entered and returned to the customer but cannot be processed fully (e.g. no recognition software in the recognition module, the note is not recognized, etc.) then a InputRefuse event will be sent and the command will complete. In this case, the output parameters will be set as follows, *noteId* = zero, *length* = zero, *orientation* = "unknown" and *signature* = NULL.

Command Message

| Payload | Type | Required |
|--------------------------------------|------|----------|
| { "timeout": 5000 integer } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|---------|------|----------|
| { | | |

```

"completionCode": "success",           string
"errorDescription": Add example to YAML, string
"errorCode": "tooManyItems",           string
"noteId": 0,                          integer
"orientation": {                      object
"frontTop": false,                   boolean
"frontBottom": false,                boolean
"backTop": false,                   boolean
"backBottom": false,                boolean
"unknown": false,                  boolean
"notSupported": false,              boolean
},
"signature": Add example to YAML      string
}

```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

"tooManyItems": There was more than one banknote inserted for creating a signature.

"noItems": There was no banknote to create a signature.

"cashInActive": A cash-in transaction is active.

"exchangeActive": The device is in the exchange state.

"positionNotEmpty": The output position is not empty so a banknote cannot be inserted.

"shutterNotOpen": Shutter failed to open.

"shutterNotClosed": Shutter failed to close.

"foreignItemsDetected": Foreign items have been detected in the input position.

noteId

Identification of note type.

orientation

Orientation of the entered banknote.

orientation/frontTop

If note is inserted wide side as the leading edge, the note was inserted with the front image facing up and the top edge of the note was inserted first. If the note is inserted short side as the leading edge, the note was inserted with the front image face up and the left edge was inserted first.

orientation/frontBottom

If note is inserted wide side as the leading edge, the note was inserted with the front image facing up and the bottom edge of the note was inserted first. If the note is inserted short side as the leading edge, the note was inserted with the front image face up and the right edge was inserted first.

orientation/backTop

If note is inserted wide side as the leading edge, the note was inserted with the back image facing up and the top edge of the note was inserted first. If the note is inserted short side as the leading edge, the note was inserted with the back image face up and the left edge was inserted first.

orientation/backBottom

If note is inserted wide side as the leading edge, the note was inserted with the back image facing up and the bottom edge of the note was inserted first. If the note is inserted short side as the leading edge, the note was inserted with the back image face up and the right edge was inserted first.

orientation/unknown

The orientation for the inserted note can not be determined.

orientation/notSupported

The hardware is not capable to determine the orientation.

signature

Base64 encoded signature data.

Event Messages

- [CashAcceptor.InputRefuseEvent](#)
- [CashAcceptor.ItemsInsertedEvent](#)
- [CashAcceptor.ItemsTakenEvent](#)
- [CashAcceptor.ItemsPresentedEvent](#)
- [CashManagement.NoteErrorEvent](#)
- [CashAcceptor.InsertItemsEvent](#)
- [CashManagement.InfoAvailableEvent](#)
- [CashAcceptor.ShutterStatusChangedEvent](#)

[6.2.20 - CashAcceptor.ConfigureNoteReader](#)

This command is used to configure the currency description configuration data into the banknote reader module. The format and location of the configuration data is vendor and/or hardware dependent.

Command Message

| Payload | Type | Required |
|---------|------|----------|
| { | | |

```
"loadAlways": false  boolean
}
```

Properties

loadAlways

If set to TRUE, the Service loads the currency description data into the note reader, even if it is already loaded.

Completion Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "exchangeActive", | string | |
| "rebootNecessary": false | boolean | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

"exchangeActive": The device is in the exchange state.

"cashInActive": A cash-in transaction is active.

"loadFailed": The load failed because the device is in a state that will not allow the configuration data to be loaded at this time, for example on some devices there may be notes present in the cash units when they should not be.

rebootNecessary

If set to TRUE, the machine needs a reboot before the note reader can be accessed again.

Event Messages

None

[6.2.21 - CashAcceptor.CompareP6Signature](#)

This command is used to compare the signatures of a reference banknote with the available signatures of the cash-in transactions.

The reference signatures are created by the CashAcceptor.CreateP6Signature command.

The transaction signatures are obtained through the CashAcceptor.GetP6Signature command.

The signatures (1 to 4) of the reference banknote are typically the signatures of the 4 orientations of the banknote.

The CashAcceptor.CompareP6Signature command may return a single indication or a list of indications to the matching signatures, each one associated to a confidence level factor. If the Service Provider does not support the confidence level factor, it returns a single indication to the best matching signature with the confidence level factor set to zero.

If the comparison completed with no matching signatures found then the command returns "ok" with *p6SignaturesIndex* empty.

This command must be used outside of the cash-in transactions and outside of exchange states.

Command Message

| Payload | Type | Required |
|---------|------|----------|
|---------|------|----------|

```
{
  "p6ReferenceSignatures": [{}],           array (object)
  "noteId": 0,                            integer
  "orientation": {},                      object
  "frontTop": false,                     boolean
  "frontBottom": false,                   boolean
  "backTop": false,                      boolean
  "backBottom": false,                   boolean
  "unknown": false,                      boolean
  "notSupported": false,                 boolean
},
"signature": Add example to YAML      string
}],
"p6Signatures": [{}],                  array (object)
}
```

See `p6ReferenceSignatures` properties.

}]

}

Properties

`p6ReferenceSignatures`

Array of P6Signature structures. Each structure represents the signature corresponding to one orientation of a single reference banknote. At least one orientation must be provided. If no orientations are provided (this array is missing or empty) the command returns an invalidData error.

`p6ReferenceSignatures/noteId`

Identification of note type.

p6ReferenceSignatures/orientation

Orientation of the entered banknote.

p6ReferenceSignatures/orientation/frontTop

If note is inserted wide side as the leading edge, the note was inserted with the front image facing up and the top edge of the note was inserted first. If the note is inserted short side as the leading edge, the note was inserted with the front image face up and the left edge was inserted first.

p6ReferenceSignatures/orientation/frontBottom

If note is inserted wide side as the leading edge, the note was inserted with the front image facing up and the bottom edge of the note was inserted first. If the note is inserted short side as the leading edge, the note was inserted with the front image face up and the right edge was inserted first.

p6ReferenceSignatures/orientation/backTop

If note is inserted wide side as the leading edge, the note was inserted with the back image facing up and the top edge of the note was inserted first. If the note is inserted short side as the leading edge, the note was inserted with the back image face up and the left edge was inserted first.

p6ReferenceSignatures/orientation/backBottom

If note is inserted wide side as the leading edge, the note was inserted with the back image facing up and the bottom edge of the note was inserted first. If the note is inserted short side as the leading edge, the note was inserted with the back image face up and the right edge was inserted first.

p6ReferenceSignatures/orientation/unknown

The orientation for the inserted note can not be determined.

p6ReferenceSignatures/orientation/notSupported

The hardware is not capable to determine the orientation.

p6ReferenceSignatures/signature

Base64 encoded signature data.

p6Signatures

Array of P6Signature structures. Each structure represents a level 2/3 signature, from the cash-in transactions, to be compared with the reference signatures in *p6ReferenceSignature*. At least one signature must be provided. If there are no signatures provided (this array is missing or empty) the command returns an invalidData error.

p6Signatures/noteId

Identification of note type.

p6Signatures/orientation

Orientation of the entered banknote.

p6Signatures/signature

Base64 encoded signature data.

Completion Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "cashInActive", | string | |
| "p6SignaturesIndex": [{ | array (object) | |
| "index": 0, | integer | |
| "confidenceLevel": 0, | integer | |
| "comparisonData": Add example to YAML | string | |
| }] | | |

}

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

"cashInActive": A cash-in transaction is active.

"exchangeActive": The device is in the exchange state.

"invalidReferenceSignature": At least one of the reference signatures is invalid. The client should prompt the operator to carefully retry the creation of the reference signatures.

"invalidTransactionSignature": At least one of the transaction signatures is invalid.

p6SignaturesIndex

Array of compare results. This array is empty when the compare operation completes with no match found. If there are matches found, *p6SignaturesIndex* contains the indexes of the matching signatures from the input parameter *p6Signatures*. If there is a match found but the Service does not support the confidence level factor, *p6SignaturesIndex* contains a single index with confidenceLevel set to zero.

p6SignaturesIndex/index

Specifies the index (zero to #*p6Signatures* - 1) of the matching signature from the input parameter *p6Signatures*.

p6SignaturesIndex/confidenceLevel

Specifies the level of confidence for the match found. This value is in a scale 1 - 100, where

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

100 is the maximum confidence level. This value is zero if the Service does not support the confidence level factor.

p6SignaturesIndex/comparisonData

Vendor dependent comparison result data. This data may be used as justification for the signature match or confidence level. This field is omitted if no additional comparison data is returned.

Event Messages

None

6.2.22 - CashAcceptor.Replenish

This command replenishes items from a single cash unit to multiple cash units. Clients can use this command to ensure that there is the optimum number of items in the cassettes by moving items from a source cash unit to a target cash unit. This is especially applicable if a replenishment cash unit is used for the replenishment and can help to minimize manual replenishment operations.

The CashAcceptor.ReplenishTarget command can be used to determine what cash units can be specified as target cash units for a given source cash unit. Any items which are removed from the source cash unit that are not of the correct currency ID and value for the target cash unit during execution of this command will be returned to the source cash unit.

The *count*, *cashInCount*, *dispensedCount* and *rejectCount* returned with the CashManagement.CashUnitInfo command will be updated as part of the execution of this command.

If the command fails after some items have been moved, the command will complete with an appropriate error code, and a CashAcceptor.IncompleteReplenishEvent will be sent.

Command Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "cashunitSource": Add example to YAML, | string | |
| "replenishTargets": [{ | array (object) | |
| "cashunitTarget": Add example to YAML, | string | |

```

"numberOfItemsToMove": 0,           integer
"removeAll": false                 boolean
}
}

```

Properties

cashunitSource

Object name of the cash unit (as stated by the [CashManagement.GetCashUnitInfo](#) command) from which items are to be removed.

replenishTargets

Array of replenish Target elements. There must be at least one array element.

replenishTargets/cashunitTarget

Object name of the cash unit (as stated by the [CashManagement.GetCashUnitInfo](#) command) to which items are to be moved.

replenishTargets/numberOfItemsToMove

The number of items to be moved to the target cash unit. Any items which are removed from the source cash unit that are not of the correct currency ID and value for the target cash unit during execution of this command will be returned to the source cash unit. This field will be ignored if the [removeAll](#) parameter is set to TRUE.

replenishTargets/removeAll

Specifies if all items are to be moved to the target cash unit. Any items which are removed from the source cash unit that are not of the correct currency ID and value for the target cash unit during execution of this command will be returned to the source cash unit. If TRUE all items in the source will be moved, regardless of the [numberOfItemsToMove](#) field value. If FALSE the number of items specified with [numberOfItemsToMove](#) will be moved.

Completion Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "cashUnitError", | string | |
| "numberOfItemsRemoved": 0, | integer | |
| "numberOfItemsRejected": 0, | integer | |
| "replenishTargetResults": [| array (object) | |
| "cashunitTarget": Add example to YAML, | string | |
| "noteID": 0, | integer | |
| "numberOfItemsReceived": 0 | integer | |
|] | | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

"cashUnitError": A problem occurred with a cash unit. A CashManagement.CashUnitErrorEvent will be sent with the details. If appropriate a IncompleteReplenishEvent will also be sent.

"invalidCashUnit": The source or target cash unit specified is invalid for this operation. The

CashAcceptor.ReplenishTarget command can be used to determine which source or target is valid.

"exchangeActive": The device is in the exchange state.

"cashInActive": A cash-in transaction is active.

numberOfItemsRemoved

Total number of items removed from the source cash unit including rejected items during execution of this command.

numberOfItemsRejected

Total number of items rejected during execution of this command.

replenishTargetResults

Array of replenishTargetResult structures. In the case where one note type has several releases and these are moved, or where items are moved from a multi denomination cash unit to a multi denomination cash unit, each target can receive several *noteID* note types. For example: If one single target was specified with the *replenishTargets* input structure, and this target received two different *noteID* note types, then the *replenishTargetResults* array will have two elements. Or if two targets were specified and the first target received two different *noteID* note types and the second target received three different *noteID* note types, then the *replenishTargetResults* array will have five elements.

replenishTargetResults/cashunitTarget

Object name of the cash unit (as stated by the [CashManagement.GetCashUnitInfo](#) command) to which items have been moved.

replenishTargetResults/noteID

Identification of note type. The note ID represents the note identifiers reported by the CashAcceptor.BanknoteTypes command.

replenishTargetResults/numberOfItemsReceived

Total number of items received in this target cash unit of the *noteID* note type. A zero value will be returned if this target cash unit did not receive any items of this note type, for

example due to a cash unit or transport jam.

Event Messages

- [CashManagement.CashUnitThresholdEvent](#)
- [CashManagement.CashUnitErrorEvent](#)
- [CashManagement.NoteErrorEvent](#)
- [CashManagement.InfoAvailableEvent](#)
- [CashAcceptor.IncompleteReplenishEvent](#)

[6.2.23 - CashAcceptor.SetCashInLimit](#)

This command specifies the amount/number of items limitation for the current cash-in transaction. This command can only be called after the `CashAcceptor.CashInStart` command and before the first `CashAcceptor.CashIn` command, otherwise it will fail with the `SequenceError` error. Any command that completes the cash-in transaction (i.e. `CashAcceptor.CashInEnd`, `CashAcceptor.CashInRollback`, `CashAcceptor.Retract` and `CashAcceptor.Reset` commands) will clear the limit.

This limit is active until the end of the current cash-in transaction. The use of this command is optional, however it needs to be called for each cash-in transaction that needs a limitation.

This command does not disable/enable the recognition of individual note types. The `CashAcceptor.ConfigureNotetypes` command must be used to refuse a certain note type during cash-in transactions.

If "limitMultiple" is specified in the `cashInLimit` capability, the command may be called multiple times to add to or override amount limits placed on the current cash-in transaction; the input parameter descriptions below define whether limits are added or overridden. If "limitMultiple" is not specified, this command can only be called once per cash-in transaction otherwise it will fail with the `SequenceError` error.

Command Message

| Payload | Type | Required |
|------------------------------------|---------|----------|
| { | | |
| "totalItemsLimit": 0, | integer | |
| "amountLimit": { | object | |
| "currencyID": Add example to YAML, | string | |

```
"amount": 0           number
}
}
```

Properties

totalItemsLimit

If set to a non-zero value, specifies a limit on the total number of items to be accepted during the cash-in transaction. If set to a zero value, this limitation will not be performed. This limitation can only be used if "limitByTotalItems" is specified in the *cashInLimit* field of the Common.Capabilities command. If *totalItemsLimit* is non-zero but not supported the UnsupportedData error will be returned and no limit will be set. This parameter overrides any previously set limit on the total number of items.

amountLimit

Array of amountLimit structures. This limitation can only be used if "limitByAmount" is reported in the *cashInLimit* field of the Common.Capabilities command. If *amountLimit* is not empty but not supported the UnsupportedData error will be returned and no limit will be set. If *amountLimit* is empty or omitted, this has no impact. If *amountLimit* is not empty, this specifies the maximum amount of the currency specified by *currencyID* which can be accepted in the current cash-in transaction. If the currency has already been specified for the current cash-in transaction, the maximum amount is overridden for that currency. If the currency has not already been specified, it is added to a set of currency specific limits to apply to the cash-in transaction. If any currency limits are specified for the current cash-in transaction, the handling of other currencies is dependent on whether the "refuseOther" flag is reported in the *cashInLimit* field of the Common.Capabilities command.

amountLimit/currencyID

Currency identifier in ISO 4217 format [Ref. 2]. This must not be three ASCII 0x20 characters.

amountLimit/amount

If set to a non-zero value, specifies a limit on the total amount of the cash-in transaction for the specified cCurrencyID. If set to a zero value, no amount limit will apply to the specified currency.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "exchangeActive" | string | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

"exchangeActive": The device is in the exchange state.

Event Messages

None

6.2.24 - CashAcceptor.CashUnitCount

This command counts the items in the cash unit(s). If it is necessary to move items internally to count them, the items should be returned to the cash unit from which they originated before completion of the command. If items could not be moved back to the cash

unit they originated from and did not get rejected, the command will complete with an appropriate error.

During the execution of this command one `CashManagement.CashUnitInfoChangedEvent` will be generated for each cash unit that has been counted successfully, or if the counts have changed, even if the overall command fails.

After completion of this command the number of items rejected can be determined by calling the `CashManagement.CashUnitInfo` command and checking the value of the `rejectCount` field within the output structure. The `rejectCount` value is incremented by one for each item rejected during execution of this command.

This command is designed to be used on devices where the `count` cannot be guaranteed to be accurate and therefore may need to be automatically counted periodically. Upon successful completion, for those cash units that have been counted, the `count` field is accurately reported with the `CashManagement.CashUnitInfo` command.

Command Message

| Payload | Type | Required |
|--|------|----------|
| { " cUNumList ": [0] array (integer) } | | |

Properties

cUNumList

Array containing the numbers of the individual cash units to be counted. If an invalid number is contained in this list, the command will fail with a `CashUnitError` error.

Completion Message

| Payload | Type | Required |
|---|--------|----------|
| { | | |
| " completionCode ": "success", | string | |
| " errorDescription<td>string</td><td></td> | string | |
| " errorCode<td>string</td><td></td> | string | |

}

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

"invalidCashUnit": At least one of the cash units specified is either invalid or does not support being counted. No cash units have been counted.

"cashInActive": A cash-in transaction is active.

"exchangeActive": The device is in the exchange state.

"tooManyItemsToCount": There were too many items. The required internal position may have been of insufficient size. All items should be returned to the cash unit from which they originated.

"countPositionNotEmpty": A required internal position is not empty so a cash unit count is not possible.

"cashUnitError": A cash unit caused a problem. A `CashManagement.CashUnitErrorEvent` will be posted with the details.

Event Messages

- [CashManagement.CashUnitInfoChangedEvent](#)
- [CashManagement.CashUnitThresholdEvent](#)
- [CashManagement.CashUnitErrorEvent](#)
- [CashManagement.NoteErrorEvent](#)
- [CashManagement.InfoAvailableEvent](#)

6.2.25 - CashAcceptor.DeviceLockControl

This command can be used to lock or unlock a CashAcceptor device, it can also be used to lock or unlock one or more cash units.

During normal device operation the device and cash units will be locked and removal will not be possible. If supported the device or cash units can be unlocked, ready for removal. In this situation the device will still remain online and cash-in or dispense operations will be possible, as long as the device or cash units are not physically removed from their normal operating position.

If the lock action is specified and the device or cash units are already locked, or if the unlock action is specified and the device or cash units are already unlocked then the action will complete successfully.

Once a cash unit has been removed and reinserted it will then have a "manualInsertion" status. This status can only be cleared by issuing a CashManagement.StartExchange/CashManagement.EndExchange command sequence.

The device and all cash units will also be locked implicitly as part of the execution of the CashManagement.EndExchange or the CashAcceptor.Reset command.

Command Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "deviceAction": "lock", | string | |
| "cashUnitAction": 0, | integer | |
| "unitLockControl": [{ | array (object) | |
| "physicalPositionName": Add example to YAML, | string | |
| "unitAction": "lock" | string | |
| }] | | |
| } | | |

Properties

deviceAction

Specifies to lock or unlock the device in its normal operating position. Following values are

possible:

"lock": Locks the device so that it cannot be removed from its normal operating position.

"unlock": Unlocks the device so that it can be removed from its normal operating position.

"noLockAction": No lock/unlock action will be performed on the device.

cashUnitAction

Specifies the type of lock/unlock action on cash units. Following values are possible:

"lockAll": Locks all cash units supported.

"unlockAll": Unlocks all cash units supported.

"lockIndividual": Locks/unlocks cash units individually as specified in the *unitLockControl* parameter.

"noLockAction":

unitLockControl

Array of UnitLockControl structures; only valid in the case where "lockIndividual" is specified in the *cashUnitAction* field. Otherwise this field will be ignored. Each element specifies one cash unit to be locked/unlocked.

unitLockControl/physicalPositionName

Specifies which cash unit is to be locked/unlocked. This name is the same as the *physicalPositionName* in the CashUnitInfo structure. Only cash units reported by the CashAcceptor.DeviceLockStatus command can be specified.

unitLockControl/unitAction

Specifies whether to lock or unlock the cash unit indicated in the *physicalPositionName* parameter. Following values are possible: "lock": Locks the specified cash unit so that it cannot be removed from the device. "unlock": Unlocks the specified cash unit so that it can be removed from the device.

Completion Message

| Payload | Type | Required |
|---------|------|----------|
|---------|------|----------|

```
{  
  "completionCode": "success",           string  
  "errorDescription": Add example to YAML,  string  
  "errorCode": "invalidCashUnit"          string  
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

"invalidCashUnit": The cash unit type specified is invalid.

"cashInActive": A cash-in transaction is active.

"exchangeActive": The device is in the exchange state.

"deviceLockFailure": The device and/or the cash units specified could not be locked/unlocked. (e.g. the lock action could not be performed because the cash unit specified to be locked had been removed).

Event Messages

- [CashManagement.CashUnitThresholdEvent](#)
- [CashManagement.CashUnitErrorEvent](#)

6.2.26 - CashAcceptor.SetMode

This command is used to set the deposit mode for the device and is only applicable for Mixed Media processing. The deposit mode determines how the device will process non cash items that are inserted. The deposit mode applies to all subsequent transactions. The deposit mode is persistent and is unaffected by a device reset by CashDispenser.Reset or reset on another interface. The command will fail with a InvalidData error where an attempt is made to set a mode that is not supported.

Command Message

| Payload | Type | Required |
|---|------|----------|
| { "mixedMode": mixedMediaNotActive string } | | |

Properties

mixedMode

Specifies the Mixed Media mode of the device. Following values are possible:

"mixedMediaNotActive": Mixed Media transactions are deactivated. This is the default mode.

"mixedMedia": Mixed Media transactions are activated in combination with the ItemProcessor interface as defined by the capability *mixedMode*.

default: mixedMediaNotActive

Completion Message

| Payload | Type | Required |
|--|------|----------|
| { "completionCode": "success", string "errorDescription": Add example to YAML, string "errorCode": "cashInActive" string } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

"cashInActive": A cash-in transaction is active.

"mediaInActive": An item processing transaction is active.

Event Messages

None

[6.2.27 - CashAcceptor.PresentMedia](#)

This command opens the shutter and presents items to be taken by the customer. The shutter is automatically closed after the media is taken. The command can be called after a CashAcceptor.CashIn, CashAcceptor.CashInRollback, CashAcceptor.Reset or CashAcceptor.CreateP6Signature command and can be used with explicit and implicit shutter control. The command is only valid on positions where *usage* reported by the CashAcceptor.PositionCapabilities command is "rollback" or "refuse" and where *presentControl* reported by the CashAcceptor.PositionCapabilities command is FALSE.

This command cannot be used to present items stacked through the CashDispenser interface. Where this is attempted the command fails with a SequenceError error.

Mixed Media Mode:

If the device is operating in Mixed Media mode (Status *mixedMode* == "mixedMedia") this command will not perform any operation unless the ItemProcessor.PresentMedia

command is called or has already been called on the ItemProcessor interface. Shutter control on devices that support Mixed Media processing is always implicit.

Command Message

| Payload | Type | Required |
|---------------------------|------|----------|
| { | | |
| "position": "null" string | | |
| } | | |

Properties

position

Describes the position where the media is to be presented. Following values are possible:

- "null": The default configuration information should be used.
- "inLeft": Present items to the left input position.
- "inRight": Present items to the right input position.
- "inCenter": Present items to of the center input position.
- "inTop": Present items to the top input position.
- "inBottom": Present items to the bottom input position.
- "inFront": Present items to the front input position.
- "inRear": Present items to the rear input position.
- "outLeft": Present items to the left output position.
- "outRight": Present items to the right output position.
- "outCenter": Present items to the center output position.
- "outTop": Present items to the top output position.
- "outBottom": Present items to the bottom output position.
- "outFront": Present items to the front output position.
- "outRear": Present items to of the rear output position.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "unsupportedPosition" | string | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

"unsupportedPosition": The position specified is not supported or is not a valid position for this command.

"shutterNotOpen": Shutter failed to open.

"noItems": There were no items to present at the specified position.

"exchangeActive": The device is in the exchange state.

"foreignItemsDetected": Foreign items have been detected in the input position.

Event Messages

- [CashAcceptor.ItemsTakenEvent](#)
- [CashAcceptor.ItemsPresentedEvent](#)

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

- [CashAcceptor.ShutterStatusChangedEvent](#)

[6.2.28 - CashAcceptor.Deplete](#)

This command removes items from multiple cash units to a single cash unit. Clients can use this command to ensure that there is the optimum number of items in the cassettes by moving items from source cash units to a target cash unit. This is especially applicable if surplus items are removed from multiple recycle cash units to a replenishment cash unit and can help to minimize manual replenishment operations.

The `CashAcceptor.DepleteSource` command can be used to determine what cash units can be specified as source cash units for a given target cash unit.

The `count`, `cashInCount`, `dispensedCount` and `rejectCount` returned with the `CashManagement.CashUnitInfo` command will be updated as part of the execution of this command.

If the command fails after some items have been moved, the command will complete with an appropriate error code, and a `CashAcceptor.IncompleteDepleteEvent` event will be sent.

Command Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| " depleteSources ": [{ | array (object) | |
| " cashunitSource ": Add example to YAML, | string | |
| " numberOfItemsToMove ": 0, | integer | |
| " removeAll<td>boolean</td><td></td> | boolean | |
| }], | | |
| " cashunitTarget ": Add example to YAML | string | |
| } | | |

Properties

`depleteSources`

Array of `DepleteSource` structures. There must be at least one element in this array.

depleteSources/cashunitSource

Object name of the cash unit (as stated by the [CashManagement.GetCashUnitInfo](#) command) from which items are to be removed.

depleteSources/numberOfItemsToMove

The number of items to be moved from the source cash unit. This must be equal to or less than the count of items reported for the cash unit specified by *numberSource*. This field will be ignored if the *removeAll* parameter is set to TRUE.

depleteSources/removeAll

Specifies if all items are to be moved from the source cash unit. If TRUE all items in the source will be moved, regardless of the *numberOfItemsToMove* field value. If FALSE the number of items specified with *numberOfItemsToMove* will be moved.

cashunitTarget

Object name of the cash unit (as stated by the [CashManagement.GetCashUnitInfo](#) command) to which items are to be moved.

Completion Message

| <u>Payload</u> | <u>Type</u> | <u>Required</u> |
|--|----------------|-----------------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "cashUnitError", | string | |
| "numberOfItemsReceived": 0, | integer | |
| "numberOfItemsRejected": 0, | integer | |
| "depleteSourceResults": [{ | array (object) | |
| "cashunitSource": Add example to YAML, | string | |
| "noteID": 0, | integer | |

```
"numberOfItemsRemoved": 0          integer  
}  
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

"cashUnitError": A problem occurred with a cash unit. A CashManagement.CashUnitErrorEvent will be sent with the details. If appropriate a CashAcceptor.IncompleteDepleteEvent will also be sent.

"invalidCashUnit": The source or target cash unit specified is invalid for this operation. The CashAcceptor.DepleteSource command can be used to determine which source or target is valid.

"cashInActive": A cash-in transaction is active.

"exchangeActive": The device is in the exchange state.

numberOfItemsReceived

Total number of items received in the target cash unit during execution of this command.

numberOfItemsRejected

Total number of items rejected during execution of this command.

depleteSourceResults

Array of DepleteSpourceResult structures. In the case where one item type has several releases and these are moved, or where items are moved from a multi denomination cash unit to a multi denomination cash unit, each source can move several *noteID* item types.

For example: If one single source was specified with the *depleteSources* input structure, and this source moved two different *noteID* item types, then the *depleteSourceResults* array will have two elements. Or if two sources were specified and the first source moved two different *noteID* item types and the second source moved three different *noteID* item types, then the *depleteSourceResults* array will have five elements.

depleteSourceResults/cashunitSource

Object name of the cash unit (as stated by the [CashManagement.GetCashUnitInfo](#) command) from which items have been removed.

depleteSourceResults/noteID

Identification of item type. The note ID represents the item identifiers reported by the [CashAcceptor.BanknoteTypes](#) command.

depleteSourceResults/numberOfItemsRemoved

Total number of items removed from this source cash unit of the *noteID* item type. A zero value will be returned if this source cash unit did not move any items of this item type, for example due to a cash unit or transport jam.

Event Messages

- [CashManagement.CashUnitThresholdEvent](#)
- [CashManagement.CashUnitErrorEvent](#)
- [CashManagement.NoteErrorEvent](#)
- [CashManagement.InfoAvailableEvent](#)
- [CashAcceptor.IncompleteDepleteEvent](#)

6.2.29 - [CashAcceptor.PreparePresent](#)

In cases where multiple bunches are to be returned under explicit shutter control, this command is used for the purpose of moving a remaining bunch to the output position explicitly before using the following commands:

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

OpenShutter

CashAcceptor.PresentMedia

The client can tell whether the additional items were left by using CashAcceptor.PresentStatus command. This command does not affect the status of the current cash-in transaction.

Command Message

| Payload | Type | Required |
|-----------------------------|------|----------|
| { | | |
| "position": "null" string | | |
| } | | |

Properties

position

Describes the position where the items are to be moved. Following values are possible:

"null": The default configuration information should be used.

"outLeft": Move items to the left output position.

"outRight": Move items to the right output position.

"outCenter": Move items to the center output position.

"outTop": Move items to the top output position.

"outBottom": Move items to the bottom output position.

"outFront": Move items to the front output position.

"outRear": Move items to the rear output position.

Completion Message

| Payload | Type | Required |
|------------------------------|--------|----------|
| { | | |
| "completionCode": "success", | string | |

```
"errorDescription": Add example to YAML, string  
"errorCode": "unsupportedPosition", string  
"position": "null" string  
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. Following values are possible:

"unsupportedPosition": The position specified is not supported or is not a valid position for this command.

"positionNotEmpty": The input or output position is not empty.

"noItems": There were no items to present at the specified position.

"cashUnitError": A cash unit caused a problem. A CashManagement.CashUnitErrorEvent will be posted with the details.

position

Describes the position where the items are to be moved. Following values are possible:

"null": The default configuration information should be used.

"outLeft": Move items to the left output position.

"outRight": Move items to the right output position.

"outCenter": Move items to the center output position.

- "outTop": Move items to the top output position.
- "outBottom": Move items to the bottom output position.
- "outFront": Move items to the front output position.
- "outRear": Move items to the rear output position.

Event Messages

- [CashManagement.CashUnitErrorEvent](#)
- [CashManagement.InfoAvailableEvent](#)

6.3 - Event Messages

6.3.1 - [CashManagement.CashUnitErrorEvent](#)

This event is generated if there is a problem with a cash unit during the execution of a command.

| Payload | Type | Required |
|--------------------------------------|---------|----------|
| { | | |
| "failure": "empty", | string | |
| "cashUnit": { | object | |
| "status": "ok", | string | |
| "type": "billCassette", | string | |
| "currencyID": Add example to YAML, | string | |
| "value": 0, | number | |
| "logicalCount": 0, | integer | |
| "maximum": 0, | integer | |
| "appLock": false, | boolean | |
| "cashUnitName": Add example to YAML, | string | |
| "initialCount": 0, | integer | |

```

"dispensedCount": 0,                                integer
"presentedCount": 0,                               integer
"retractedCount": 0,                               integer
"rejectCount": 0,                                 integer
"minimum": 0,                                    integer
"physicalPositionName": Add example to YAML,    string
"unitID": Add example to YAML,                   string
"count": 0,                                       integer
"maximumCapacity": 0,                            integer
"hardwareSensor": false,                         boolean
"itemType": {                                     object
  "all": false,                                  boolean
  "unfit": false,                                boolean
  "individual": false,                           boolean
  "level1": false,                                boolean
  "level2": false,                                boolean
  "level3": false,                                boolean
  "itemProcessor": false,                         boolean
  "unfitIndividual": false,                      boolean
},
"cashInCount": 0,                                 integer
"noteNumberList": {                                object
  "noteNumber": [{                                array (object)
    "noteID": 0,                                 integer
    "count": 0,                                 integer
  }]
}

```

```

},
"noteIDs": [0]                                array (integer)
}
}

```

Properties

failure

Specifies the kind of failure that occurred in the cash unit. Following values are possible:

- empty - Specified cash unit is empty.
- error - Specified cash unit has malfunctioned.
- full - Specified cash unit is full.
- locked - Specified cash unit is locked.
- invalid - Specified cash unit is invalid.
- config - An attempt has been made to change the settings of a self-configuring cash unit.
- notConfigured - Specified cash unit is not configured.

cashUnit

The cash unit object that caused the problem.

cashUnit/status

Supplies the status of the cash unit. Following values are possible:

- ok - The cash unit is in a good state.
- full - The cash unit is full.
- high - The cash unit is almost full (i.e. reached or exceeded the threshold defined by *maximum*).
- low - The cash unit is almost empty (i.e. reached or below the threshold defined by *minimum*).
- empty - The cash unit is empty, or insufficient items in the cash unit are preventing further dispense operations.
- inoperative - The cash unit is inoperative.
- missing - The cash unit is missing.
- noValue - The values of the specified cash unit are not available.
- noReference - There is no reference value available for the notes in this cash unit.

The cash unit has not been calibrated.

- manipulated - The cash unit has been inserted (including removal followed by a reinsertion) when the device was not in the exchange state. This cash unit cannot be dispensed from.

cashUnit/type

Type of cash unit. Following values are possible:

- notApplicable - Not applicable. Typically means cash unit is missing.
- rejectCassette - Reject cash unit. This type will also indicate a combined reject/retract cash unit.
- billCassette - Cash unit containing bills.
- coinCylinder - Coin cylinder.
- coinDispenser - Coin dispenser as a whole unit.
- retractCassette - Retract cash unit.
- coupon - Cash unit containing coupons or advertising material.
- document - Cash unit containing documents.
- replenishmentContainer - Replenishment container. A cash unit can be refilled from a replenishment container.
- recycling - Recycling cash unit. This unit is only present when the device implements the Dispenser and CashAcceptor interfaces.
- cashIn - Cash-in cash unit.

cashUnit/currencyID

A three character string storing the ISO format [Ref. 2] Currency ID. This value will be omitted for cash units which contain items of more than one currency type or items to which currency is not applicable. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

cashUnit/value

Supplies the value of a single item in the cash unit. This value is expressed as floating point value. If the *currencyID* field for this cash unit is omitted, then this field will contain zero. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

cashUnit/logicalCount

The meaning of this count depends on the type of cash unit. This value is persistent. For all cash units except retract cash units (*type* is not *retractCassette*) this value specifies the number of items inside the cash unit. For all dispensing cash units (*type* is *billCassette*,

coinCylinder, coinDispenser, coupon, document or recycling), this value includes any items from the cash unit not yet presented to the customer. This count is only decremented when the items are either known to be in customer access or successfully rejected. If the cash unit is usable from the CashAcceptor interface (*type* is *recycling*, *cashIn*, *retractCassette* or *rejectCassette*) then this value will be incremented as a result of a cash-in operation. Note that for a reject cash unit (*type* is *rejectCassette*), this value is unreliable, since the typical reason for dumping items to the reject cash unit is a suspected count failure. For a retract cash unit (*type* is *retractCassette*) this value specifies the number of retract operations which result in items entering the cash unit.

cashUnit/maximum

When *count* reaches this value the threshold event

`CashManagement.CashUnitThresholdEvent (high)` will be generated. This value can be different from the actual capacity of the cassette. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

cashUnit/appLock

If this value is TRUE items cannot be dispensed from or deposited into the cash unit. If this value is TRUE and the client attempts to use the cash unit a `CashManagement.CashUnitErrorEvent` event will be generated and an error completion message will be returned. This value is persistent.

cashUnit/cashUnitName

A name which helps to identify the type of the cash unit. This is especially useful in the case of cash units of type *document* where different documents can have the same currency and value. For example, travelers checks and bank checks may have the same currency and value but still need to be identifiable as different types of document. Where this value is not relevant (e.g. in bill cash units) the property can be omitted. This value is persistent.

cashUnit/initialCount

Initial number of items contained in the cash unit. This value is persistent.

cashUnit/dispensedCount

The number of items dispensed from this cash unit. This count is incremented when the items are removed from the cash units. This count includes any items that were rejected during the dispense operation and are no longer in this cash unit. This field is always zero

for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

cashUnit/presentedCount

The number of items from this cash unit that have been presented to the customer. This count is incremented when the items are presented to the customer. If it is unknown if a customer has been presented with the items, then this count is not updated. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

cashUnit/retractedCount

The number of items that have been accessible to a customer and retracted into the cash unit. This value is persistent.

cashUnit/rejectCount

The number of items dispensed from this cash unit which have been rejected, are in a cash unit other than this cash unit, and which have not been accessible to a customer. This value may be unreliable, since a typical reason for rejecting items is a suspected pick failure. Other reasons for rejecting items may include incorrect note denominations, classifications not valid for dispensing, or where the transaction has been cancelled and a Reject command has been called. For reject and retract cash units (*type* is *rejectCassette* or *retractCassette*) this field does not apply and will be reported as zero. This value is persistent.

cashUnit/minimum

This field is not applicable to retract and reject cash units. For all cash units which dispense items (all other), when *count* reaches this value the threshold event CashManagement.CashUnitThresholdEvent (*low*) will be generated. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

cashUnit/physicalPositionName

A name identifying the physical location of the cash unit.

cashUnit/unitID

A 5 character string uniquely identifying the cash unit.

cashUnit/count

As defined by the *logicalCount* description, but with the following exceptions: This count does not include items dispensed but not yet presented. On cash units with *type* set to "retractCassette" the count represents the number of items, unless the device cannot count items during a retract, in which case this count will be zero. This value is persistent.

cashUnit/maximumCapacity

The maximum number of items the cash unit can hold. This is only for informational purposes. No threshold event CashManagement.CashUnitThresholdEvent will be generated. This value is persistent.

cashUnit/hardwareSensor

Specifies whether or not threshold events can be generated based on hardware sensors in the device. If this value is TRUE then threshold events may be generated based on hardware sensors as opposed to counts.

cashUnit/itemType

Specifies the type of items the cash unit takes as a combination of the following flags. The table in the Comments section of this command defines how to interpret the combination of these flags (TODO: include Table)

cashUnit/itemType/all

The cash unit takes all fit banknote types. These are level 4 notes which are fit for recycling.

cashUnit/itemType/unfit

The cash unit takes all unfit banknotes. These are level 4 notes which are unfit for recycling.

cashUnit/itemType/individual

The cash unit takes all types of fit banknotes specified in an individual list. These are level 4 notes which are fit for recycling.

cashUnit/itemType/level1

Level 1 note types are stored in this cash unit.

cashUnit/itemType/level2

If notes can be classified as level 2, then level 2 note types are stored in this cash unit.

cashUnit/itemType/level3

If notes can be classified as level 3, then level 3 note types are stored in this cash unit.

cashUnit/itemType/itemProcessor

The cash unit can accept items on the ItemProcessor interface.

cashUnit/itemType/unfitIndividual

The cash unit takes all types of unfit banknotes specified in an individual list. These are level 4 notes which are unfit for recycling.

cashUnit/cashInCount

Count of items that have entered the cash unit. This counter is incremented whenever an item enters a cash unit for any reason, unless it originated from this cash unit but was returned without being accessible to a customer. For a retract cash unit this value represents the total number of items of all types in the cash unit, or if the device cannot count items during a retract operation this value will be zero. This value is persistent.

cashUnit/noteNumberList

Array of cash items inside the cash unit. The content of this structure is persistent. If the cash unit is Dispenser specific cash unit with *type billCassette* or the contents of the cash unit are not known this structure will be omitted. If the cash unit is of *type retractCassette* this pointer will be omitted except for the following cases:

- If the retract cash unit is configured to accept level 2 notes then the number and type of level 2 notes is returned in the *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of level 2 notes.
- If items are recognized during retract operations then the number and type of notes

retracted is returned in *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of retracted items.

`cashUnit/noteNumberList/noteNumber`

Array of banknote numbers the cash unit contains.

`cashUnit/noteNumberList/noteNumber/noteID`

Identification of note type. The Note ID represents the note identifiers reported by the *CashAcceptor.BanknoteTypes* command. If this value is zero then the note type is unknown.

`cashUnit/noteNumberList/noteNumber/count`

Actual count of cash items. The value is incremented each time cash items are moved to a cash unit. In the case of recycle cash units this count is decremented as defined in the description of the *logicalCount* field.

`cashUnit/noteIDs`

Array of integers which contains the note IDs of the banknotes the cash-in cash unit or recycle cash unit can take. This field only applies to *individual* cassette types. If there are no note IDs defined for the cassette or the cassette is not defined as *individual* then *noteIDs* will be omitted.

6.3.2 - `CashAcceptor.InputRefuseEvent`

This event specifies that the device has refused either a portion or the entire amount of the cash-in order.

| Payload | Type | Required |
|----------------------------|--------|----------|
| { | | |
| "reason": "cashInUnitFull" | string | |
| } | | |

Properties

reason

Reason for refusing a part of the amount. Following values are possible:

"cashInUnitFull": Cash unit is full.

"invalidBill": Recognition of the items took place, but one or more of the items are invalid.

"noBillsToDeposit": There are no items in the input area.

"depositFailure": A deposit has failed for a reason not covered by the other reasons and the failure is not a fatal hardware problem, for example failing to pick an item from the input area.

"commonInputComponentFailure": Failure of a common input component which is shared by all cash units.

"stackerFull": The intermediate stacker is full.

"foreignItemsDetected": Foreign items have been detected in the input position.

"invalidBunch": Recognition of the items did not take place. The bunch of notes inserted is invalid, e.g. it is too large or was inserted incorrectly.

"counterfeit": One or more counterfeit items have been detected and refused. This is only applicable where notes are not classified as level 2 and the device is capable of differentiating between invalid and counterfeit items.

"limitOverTotalItems": Number of items count exceeded the limitation set with the CashAcceptor.SetCashInLimit command.

"limitOverAmount": Amount exceeded the limitation set with the CashAcceptor.SetCashInLimit command.

[6.3.3 - CashManagement.NoteErrorEvent](#)

This event specifies the reason for a note detection error during the execution of a command.

| Payload | Type | Required |
|------------------------|--------|----------|
| { | | |
| "reason": "doubleNote" | string | |
| } | | |

Properties

reason

The reason for the notes detection error. Following values are possible:

- doubleNote - Double notes have been detected.
- longNote - A long note has been detected.
- skewedNote - A skewed note has been detected.
- incorrectCount - An item counting error has occurred.
- notesTooClose - Notes have been detected as being too close.
- otherNoteError - An item error not covered by the other values has been detected.
- shortNote - Short notes have been detected.

[6.3.4 - CashAcceptor.SubCashInEvent](#)

This event is generated when one of the sub cash-in operations into which the cash-in operation was divided has finished successfully.

| Payload | Type | Required |
|------------------|----------------|----------|
| { | | |
| "noteNumber": [{ | array (object) | |
| "noteID": 0, | integer | |
| "count": 0 | integer | |
| }] | | |
| } | | |

Properties

noteNumber

Array of banknote numbers the cash unit contains.

noteNumber/noteID

Identification of note type. The Note ID represents the note identifiers reported by the *CashAcceptor.BanknoteTypes* command. If this value is zero then the note type is unknown.

noteNumber/count

Actual count of cash items. The value is incremented each time cash items are moved to a cash unit. In the case of recycle cash units this count is decremented as defined in the description of the *logicalCount* field.

6.3.5 - CashManagement.InfoAvailableEvent

This execute event is generated when information is available for items detected during the cash processing operation.

| Payload | Type | Required |
|-----------------------|----------------|----------|
| { | | |
| "itemInfoSummary": [{ | array (object) | |
| "level": "level1", | string | |
| "numOfItems": 0 | integer | |
| }] | | |
| } | | |

Properties**itemInfoSummary**

Array of itemInfoSummary objects, one object for every level.

itemInfoSummary/level

Defines the note level. Following values are possible:

- level1 - Information for level 1 notes.
- level2 - Information for level 2 notes.
- level3 - Information for level 3 notes.
- level4 - Information for level 4 notes.

itemInfoSummary/numOfItems

Number of items classified as *level* which have information available.

6.3.6 - CashAcceptor.InsertItemsEvent

This event notifies the client when the device is ready for the user to insert items.

6.3.7 - CashManagement.CashUnitThresholdEvent

This user event is generated when a threshold condition has occurred in one of the cash units.

This event can be triggered either by hardware sensors in the device or by the **count** reaching the **minimum** or **maximum** value as specified in the GetCashUnitInfo structure.

The client can check if the device has hardware sensors by querying the **hardwareSensor** field of the cash unit structure. If a cash unit has this capability then threshold events based on hardware sensors will be triggered if the **maximum** or **minimum** values are not used or are set to zero.

| Payload | Type | Required |
|--------------------------------------|---------|----------|
| { | | |
| "status": "ok", | string | |
| "type": "billCassette", | string | |
| "currencyID": Add example to YAML, | string | |
| "value": 0, | number | |
| "logicalCount": 0, | integer | |
| "maximum": 0, | integer | |
| "appLock": false, | boolean | |
| "cashUnitName": Add example to YAML, | string | |
| "initialCount": 0, | integer | |
| "dispensedCount": 0, | integer | |
| "presentedCount": 0, | integer | |
| "retractedCount": 0, | integer | |
| "rejectCount": 0, | integer | |

| | |
|--|-----------------|
| "minimum": 0, | integer |
| "physicalPositionName": Add example to YAML, | string |
| "unitID": Add example to YAML, | string |
| "count": 0, | integer |
| "maximumCapacity": 0, | integer |
| "hardwareSensor": false, | boolean |
| "itemType": { | object |
| "all": false, | boolean |
| "unfit": false, | boolean |
| "individual": false, | boolean |
| "level1": false, | boolean |
| "level2": false, | boolean |
| "level3": false, | boolean |
| "itemProcessor": false, | boolean |
| "unfitIndividual": false | boolean |
| }, | |
| "cashInCount": 0, | integer |
| "noteNumberList": { | object |
| "noteNumber": [{ | array (object) |
| "noteID": 0, | integer |
| "count": 0 | integer |
| }] | |
| }, | |
| "noteIDs": [0] | array (integer) |
| } | |

Properties

status

Supplies the status of the cash unit. Following values are possible:

- ok - The cash unit is in a good state.
- full - The cash unit is full.
- high - The cash unit is almost full (i.e. reached or exceeded the threshold defined by *maximum*).
- low - The cash unit is almost empty (i.e. reached or below the threshold defined by *minimum*).
- empty - The cash unit is empty, or insufficient items in the cash unit are preventing further dispense operations.
- inoperative - The cash unit is inoperative.
- missing - The cash unit is missing.
- noValue - The values of the specified cash unit are not available.
- noReference - There is no reference value available for the notes in this cash unit. The cash unit has not been calibrated.
- manipulated - The cash unit has been inserted (including removal followed by a reinsertion) when the device was not in the exchange state. This cash unit cannot be dispensed from.

type

Type of cash unit. Following values are possible:

- notApplicable - Not applicable. Typically means cash unit is missing.
- rejectCassette - Reject cash unit. This type will also indicate a combined reject/retract cash unit.
- billCassette - Cash unit containing bills.
- coinCylinder - Coin cylinder.
- coinDispenser - Coin dispenser as a whole unit.
- retractCassette - Retract cash unit.
- coupon - Cash unit containing coupons or advertising material.
- document - Cash unit containing documents.
- replenishmentContainer - Replenishment container. A cash unit can be refilled from a replenishment container.
- recycling - Recycling cash unit. This unit is only present when the device implements the Dispenser and CashAcceptor interfaces.
- cashIn - Cash-in cash unit.

currencyID

A three character string storing the ISO format [Ref. 2] Currency ID. This value will be

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

omitted for cash units which contain items of more than one currency type or items to which currency is not applicable. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

value

Supplies the value of a single item in the cash unit. This value is expressed as floating point value. If the *currencyID* field for this cash unit is omitted, then this field will contain zero. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

logicalCount

The meaning of this count depends on the type of cash unit. This value is persistent. For all cash units except retract cash units (*type* is not *retractCassette*) this value specifies the number of items inside the cash unit. For all dispensing cash units (*type* is *billCassette*, *coinCylinder*, *coinDispenser*, *coupon*, *document* or *recycling*), this value includes any items from the cash unit not yet presented to the customer. This count is only decremented when the items are either known to be in customer access or successfully rejected. If the cash unit is usable from the CashAcceptor interface (*type* is *recycling*, *cashIn*, *retractCassette* or *rejectCassette*) then this value will be incremented as a result of a cash-in operation. Note that for a reject cash unit (*type* is *rejectCassette*), this value is unreliable, since the typical reason for dumping items to the reject cash unit is a suspected count failure. For a retract cash unit (*type* is *retractCassette*) this value specifies the number of retract operations which result in items entering the cash unit.

maximum

When *count* reaches this value the threshold event *CashManagement.CashUnitThresholdEvent* (*high*) will be generated. This value can be different from the actual capacity of the cassette. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

appLock

If this value is TRUE items cannot be dispensed from or deposited into the cash unit. If this value is TRUE and the client attempts to use the cash unit a *CashManagement.CashUnitErrorEvent* event will be generated and an error completion message will be returned. This value is persistent.

cashUnitName

A name which helps to identify the type of the cash unit. This is especially useful in the case of cash units of type *document* where different documents can have the same currency and value. For example, travelers checks and bank checks may have the same currency and value but still need to be identifiable as different types of document. Where this value is not relevant (e.g. in bill cash units) the property can be omitted. This value is persistent.

initialCount

Initial number of items contained in the cash unit. This value is persistent.

dispensedCount

The number of items dispensed from this cash unit. This count is incremented when the items are removed from the cash units. This count includes any items that were rejected during the dispense operation and are no longer in this cash unit. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

presentedCount

The number of items from this cash unit that have been presented to the customer. This count is incremented when the items are presented to the customer. If it is unknown if a customer has been presented with the items, then this count is not updated. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

retractedCount

The number of items that have been accessible to a customer and retracted into the cash unit. This value is persistent.

rejectCount

The number of items dispensed from this cash unit which have been rejected, are in a cash unit other than this cash unit, and which have not been accessible to a customer. This value may be unreliable, since a typical reason for rejecting items is a suspected pick failure. Other reasons for rejecting items may include incorrect note denominations, classifications not valid for dispensing, or where the transaction has been cancelled and a Reject command has been called. For reject and retract cash units (*type* is *rejectCassette* or *retractCassette*) this field does not apply and will be reported as zero. This value is persistent.

minimum

This field is not applicable to retract and reject cash units. For all cash units which dispense items (all other), when *count* reaches this value the threshold event CashManagement.CashUnitThresholdEvent (*low*) will be generated. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

physicalPositionName

A name identifying the physical location of the cash unit.

unitID

A 5 character string uniquely identifying the cash unit.

count

As defined by the *logicalCount* description, but with the following exceptions: This count does not include items dispensed but not yet presented. On cash units with *type* set to "retractCassette" the count represents the number of items, unless the device cannot count items during a retract, in which case this count will be zero. This value is persistent.

maximumCapacity

The maximum number of items the cash unit can hold. This is only for informational purposes. No threshold event CashManagement.CashUnitThresholdEvent will be generated. This value is persistent.

hardwareSensor

Specifies whether or not threshold events can be generated based on hardware sensors in the device. If this value is TRUE then threshold events may be generated based on hardware sensors as opposed to counts.

itemType

Specifies the type of items the cash unit takes as a combination of the following flags. The table in the Comments section of this command defines how to interpret the combination

of these flags (TODO: include Table)

itemType/all

The cash unit takes all fit banknote types. These are level 4 notes which are fit for recycling.

itemType/unfit

The cash unit takes all unfit banknotes. These are level 4 notes which are unfit for recycling.

itemType/individual

The cash unit takes all types of fit banknotes specified in an individual list. These are level 4 notes which are fit for recycling.

itemType/level1

Level 1 note types are stored in this cash unit.

itemType/level2

If notes can be classified as level 2, then level 2 note types are stored in this cash unit.

itemType/level3

If notes can be classified as level 3, then level 3 note types are stored in this cash unit.

itemType/itemProcessor

The cash unit can accept items on the ItemProcessor interface.

itemType/unfitIndividual

The cash unit takes all types of unfit banknotes specified in an individual list. These are level 4 notes which are unfit for recycling.

cashInCount

Count of items that have entered the cash unit. This counter is incremented whenever an item enters a cash unit for any reason, unless it originated from this cash unit but was returned without being accessible to a customer. For a retract cash unit this value represents the total number of items of all types in the cash unit, or if the device cannot count items during a retract operation this value will be zero. This value is persistent.

noteNumberList

Array of cash items inside the cash unit. The content of this structure is persistent. If the cash unit is Dispenser specific cash unit with *type billCassette* or the contents of the cash unit are not known this structure will be omitted. If the cash unit is of *type retractCassette* this pointer will be omitted except for the following cases:

- If the retract cash unit is configured to accept level 2 notes then the number and type of level 2 notes is returned in the *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of level 2 notes.
- If items are recognized during retract operations then the number and type of notes retracted is returned in *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of retracted items.

noteNumberList/noteNumber

Array of banknote numbers the cash unit contains.

noteNumberList/noteNumber/noteID

Identification of note type. The Note ID represents the note identifiers reported by the *CashAcceptor.BanknoteTypes* command. If this value is zero then the note type is unknown.

noteNumberList/noteNumber/count

Actual count of cash items. The value is incremented each time cash items are moved to a cash unit. In the case of recycle cash units this count is decremented as defined in the description of the *logicalCount* field.

noteIDs

Array of integers which contains the note IDs of the banknotes the cash-in cash unit or recycle cash unit can take. This field only applies to *individual* cassette types. If there are no note IDs defined for the cassette or the cassette is not defined as *individual* then *noteIDs* will be omitted.

6.3.8 - [CashManagement.CashUnitInfoChangedEvent](#)

This service event is generated under the following circumstances:

- It is generated whenever [status](#) changes. For instance, a cash unit has been removed or inserted, or a cash unit has become empty or full.
- This event will also be generated for every cash unit changed in any way (including changes to counts, e.g. [count](#), [rejectCount](#), [initialCount](#), [dispensedCount](#) and [presentedCount](#)) as a result of the [CashManagement.SetCashUnitInfo](#) command.
- This event will also be fired when any change is made to a cash unit by the following commands, except for changes to counts (e.g. [count](#), [rejectCount](#), [initialCount](#), [dispensedCount](#) and [presentedCount](#)):

[Dispenser.CalibrateCashUnit](#)

[Dispenser.TestCashUnit](#)

- In addition this event will be generated when a cash unit has been counted during the [CashAcceptor.CashUnitCount](#) command execution.

When a cash unit is removed, the status of the cash unit becomes missing.

If a new cash unit is inserted the cash unit structure reported by the last [CashManagement.GetCashUnitInfo](#) command is no longer valid. In that case a client should issue a [CashManagement.GetCashUnitInfo](#) command after receiving this event to obtain updated cash unit information.

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "status": "ok", | string | |
| "type": "billCassette", | string | |
| "currencyID": Add example to YAML, | string | |
| "value": 0, | number | |
| "logicalCount": 0, | integer | |
| "maximum": 0, | integer | |
| "appLock": false, | boolean | |
| "cashUnitName": Add example to YAML, | string | |
| "initialCount": 0, | integer | |

```

"dispensedCount": 0,                                integer
"presentedCount": 0,                               integer
"retractedCount": 0,                               integer
"rejectCount": 0,                                 integer
"minimum": 0,                                    integer
"physicalPositionName": Add example to YAML,    string
"unitID": Add example to YAML,                   string
"count": 0,                                       integer
"maximumCapacity": 0,                            integer
"hardwareSensor": false,                         boolean
"itemType": {                                     object
  "all": false,                                  boolean
  "unfit": false,                                boolean
  "individual": false,                           boolean
  "level1": false,                                boolean
  "level2": false,                                boolean
  "level3": false,                                boolean
  "itemProcessor": false,                         boolean
  "unfitIndividual": false,                      boolean
},
"cashInCount": 0,                                 integer
"noteNumberList": {                                object
  "noteNumber": [{                                array (object)
    "noteID": 0,                                 integer
    "count": 0,                                 integer
  }]
}

```

```

},
"noteIDs": [0]                                array (integer)
}

```

Properties

status

Supplies the status of the cash unit. Following values are possible:

- ok - The cash unit is in a good state.
- full - The cash unit is full.
- high - The cash unit is almost full (i.e. reached or exceeded the threshold defined by *maximum*).
- low - The cash unit is almost empty (i.e. reached or below the threshold defined by *minimum*).
- empty - The cash unit is empty, or insufficient items in the cash unit are preventing further dispense operations.
- inoperative - The cash unit is inoperative.
- missing - The cash unit is missing.
- noValue - The values of the specified cash unit are not available.
- noReference - There is no reference value available for the notes in this cash unit. The cash unit has not been calibrated.
- manipulated - The cash unit has been inserted (including removal followed by a reinsertion) when the device was not in the exchange state. This cash unit cannot be dispensed from.

type

Type of cash unit. Following values are possible:

- notApplicable - Not applicable. Typically means cash unit is missing.
- rejectCassette - Reject cash unit. This type will also indicate a combined reject/retract cash unit.
- billCassette - Cash unit containing bills.
- coinCylinder - Coin cylinder.
- coinDispenser - Coin dispenser as a whole unit.
- retractCassette - Retract cash unit.
- coupon - Cash unit containing coupons or advertising material.
- document - Cash unit containing documents.
- replenishmentContainer - Replenishment container. A cash unit can be refilled from

a replenishment container.

- recycling - Recycling cash unit. This unit is only present when the device implements the Dispenser and CashAcceptor interfaces.
- cashIn - Cash-in cash unit.

currencyID

A three character string storing the ISO format [Ref. 2] Currency ID. This value will be omitted for cash units which contain items of more than one currency type or items to which currency is not applicable. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

value

Supplies the value of a single item in the cash unit. This value is expressed as floating point value. If the *currencyID* field for this cash unit is omitted, then this field will contain zero. If the *status* field for this cash unit is *noValue* it is the responsibility of the client to assign a value to this field. This value is persistent.

logicalCount

The meaning of this count depends on the type of cash unit. This value is persistent. For all cash units except retract cash units (*type* is not *retractCassette*) this value specifies the number of items inside the cash unit. For all dispensing cash units (*type* is *billCassette*, *coinCylinder*, *coinDispenser*, *coupon*, *document* or *recycling*), this value includes any items from the cash unit not yet presented to the customer. This count is only decremented when the items are either known to be in customer access or successfully rejected. If the cash unit is usable from the CashAcceptor interface (*type* is *recycling*, *cashIn*, *retractCassette* or *rejectCassette*) then this value will be incremented as a result of a cash-in operation. Note that for a reject cash unit (*type* is *rejectCassette*), this value is unreliable, since the typical reason for dumping items to the reject cash unit is a suspected count failure. For a retract cash unit (*type* is *retractCassette*) this value specifies the number of retract operations which result in items entering the cash unit.

maximum

When *count* reaches this value the threshold event

CashManagement.CashUnitThresholdEvent (*high*) will be generated. This value can be different from the actual capacity of the cassette. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

appLock

If this value is TRUE items cannot be dispensed from or deposited into the cash unit. If this value is TRUE and the client attempts to use the cash unit a `CashManagement.CashUnitErrorEvent` event will be generated and an error completion message will be returned. This value is persistent.

cashUnitName

A name which helps to identify the type of the cash unit. This is especially useful in the case of cash units of type *document* where different documents can have the same currency and value. For example, travelers checks and bank checks may have the same currency and value but still need to be identifiable as different types of document. Where this value is not relevant (e.g. in bill cash units) the property can be omitted. This value is persistent.

initialCount

Initial number of items contained in the cash unit. This value is persistent.

dispensedCount

The number of items dispensed from this cash unit. This count is incremented when the items are removed from the cash units. This count includes any items that were rejected during the dispense operation and are no longer in this cash unit. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

presentedCount

The number of items from this cash unit that have been presented to the customer. This count is incremented when the items are presented to the customer. If it is unknown if a customer has been presented with the items, then this count is not updated. This field is always zero for cash units with a *type* of *rejectCassette* or *retractCassette*. This value is persistent.

retractedCount

The number of items that have been accessible to a customer and retracted into the cash unit. This value is persistent.

rejectCount

The number of items dispensed from this cash unit which have been rejected, are in a cash

unit other than this cash unit, and which have not been accessible to a customer. This value may be unreliable, since a typical reason for rejecting items is a suspected pick failure. Other reasons for rejecting items may include incorrect note denominations, classifications not valid for dispensing, or where the transaction has been cancelled and a Reject command has been called. For reject and retract cash units (*type* is *rejectCassette* or *retractCassette*) this field does not apply and will be reported as zero. This value is persistent.

minimum

This field is not applicable to retract and reject cash units. For all cash units which dispense items (all other), when *count* reaches this value the threshold event CashManagement.CashUnitThresholdEvent (*low*) will be generated. If this value is non-zero then hardware sensors in the device do not trigger threshold events. If this value is zero then hardware sensors will trigger threshold events if *hardwareSensor* is TRUE. This value is persistent.

physicalPositionName

A name identifying the physical location of the cash unit.

unitID

A 5 character string uniquely identifying the cash unit.

count

As defined by the *logicalCount* description, but with the following exceptions: This count does not include items dispensed but not yet presented. On cash units with *type* set to "retractCassette" the count represents the number of items, unless the device cannot count items during a retract, in which case this count will be zero. This value is persistent.

maximumCapacity

The maximum number of items the cash unit can hold. This is only for informational purposes. No threshold event CashManagement.CashUnitThresholdEvent will be generated. This value is persistent.

hardwareSensor

Specifies whether or not threshold events can be generated based on hardware sensors in

the device. If this value is TRUE then threshold events may be generated based on hardware sensors as opposed to counts.

itemType

Specifies the type of items the cash unit takes as a combination of the following flags. The table in the Comments section of this command defines how to interpret the combination of these flags (TODO: include Table)

itemType/all

The cash unit takes all fit banknote types. These are level 4 notes which are fit for recycling.

itemType/unfit

The cash unit takes all unfit banknotes. These are level 4 notes which are unfit for recycling.

itemType/individual

The cash unit takes all types of fit banknotes specified in an individual list. These are level 4 notes which are fit for recycling.

itemType/level1

Level 1 note types are stored in this cash unit.

itemType/level2

If notes can be classified as level 2, then level 2 note types are stored in this cash unit.

itemType/level3

If notes can be classified as level 3, then level 3 note types are stored in this cash unit.

itemType/itemProcessor

The cash unit can accept items on the ItemProcessor interface.

itemType/unfitIndividual

The cash unit takes all types of unfit banknotes specified in an individual list. These are level 4 notes which are unfit for recycling.

cashInCount

Count of items that have entered the cash unit. This counter is incremented whenever an item enters a cash unit for any reason, unless it originated from this cash unit but was returned without being accessible to a customer. For a retract cash unit this value represents the total number of items of all types in the cash unit, or if the device cannot count items during a retract operation this value will be zero. This value is persistent.

noteNumberList

Array of cash items inside the cash unit. The content of this structure is persistent. If the cash unit is Dispenser specific cash unit with *type billCassette* or the contents of the cash unit are not known this structure will be omitted. If the cash unit is of *type retractCassette* this pointer will be omitted except for the following cases:

- If the retract cash unit is configured to accept level 2 notes then the number and type of level 2 notes is returned in the *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of level 2 notes.
- If items are recognized during retract operations then the number and type of notes retracted is returned in *noteNumberList* and *count* contains the number of retract operations. *cashInCount* contains the actual number of retracted items.

noteNumberList/noteNumber

Array of banknote numbers the cash unit contains.

noteNumberList/noteNumber/noteID

Identification of note type. The Note ID represents the note identifiers reported by the *CashAcceptor.BanknoteTypes* command. If this value is zero then the note type is unknown.

noteNumberList/noteNumber/count

Actual count of cash items. The value is incremented each time cash items are moved to a cash unit. In the case of recycle cash units this count is decremented as defined in the description of the *logicalCount* field.

noteIDs

Array of integers which contains the note IDs of the banknotes the cash-in cash unit or recycle cash unit can take. This field only applies to *individual* cassette types. If there are no note IDs defined for the cassette or the cassette is not defined as *individual* then *noteIDs* will be omitted.

6.3.9 - CashAcceptor.IncompleteReplenishEvent

This event is generated when some items had been moved before the CashAcceptor.Replenish command failed with an error code (not "success"), but some items were moved then the details will be reported with this event. This event can only occur once per command.

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "replenish": { | object | |
| "numberOfItemsRemoved": 0, | integer | |
| "numberOfItemsRejected": 0, | integer | |
| "replenishTargetResults": [{}] | array (object) | |
| "cashunitTarget": Add example to YAML, | string | |
| "noteID": 0, | integer | |
| "numberOfItemsReceived": 0 | integer | |
| }] | | |
| } | | |
| } | | |

Properties

replenish

Note that in this case the values in this structure report the amount and number of each denomination that have actually been moved during the replenishment command.

replenish/numberOfItemsRemoved

Total number of items removed from the source cash unit including rejected items during execution of this command.

replenish/numberOfItemsRejected

Total number of items rejected during execution of this command.

replenish/replenishTargetResults

Array of replenishTargetResult structures. In the case where one note type has several releases and these are moved, or where items are moved from a multi denomination cash unit to a multi denomination cash unit, each target can receive several *noteID* note types. For example: If one single target was specified with the *replenishTargets* input structure, and this target received two different *noteID* note types, then the *replenishTargetResults* array will have two elements. Or if two targets were specified and the first target received two different *noteID* note types and the second target received three different *noteID* note types, then the *replenishTargetResults* array will have five elements.

replenish/replenishTargetResults/cashunitTarget

Object name of the cash unit (as stated by the [CashManagement.GetCashUnitInfo](#) command) to which items have been moved.

replenish/replenishTargetResults/noteID

Identification of note type. The note ID represents the note identifiers reported by the [CashAcceptor.BanknoteTypes](#) command.

replenish/replenishTargetResults/numberOfItemsReceived

Total number of items received in this target cash unit of the *noteID* note type. A zero value will be returned if this target cash unit did not receive any items of this note type, for example due to a cash unit or transport jam.

6.3.10 - CashAcceptor.IncompleteDepleteEvent

This execute event is generated when some items had been moved before the CashAcceptor.Deplete command failed with an error code (not "success"), but some items were moved. In this case the details will be reported with this event. This event can only occur once per command.

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "deplete": { | object | |
| "numberOfItemsReceived": 0, | integer | |
| "numberOfItemsRejected": 0, | integer | |
| "depleteSourceResults": [{ | array (object) | |
| "cashunitSource": Add example to YAML, | string | |
| "noteID": 0, | integer | |
| "numberOfItemsRemoved": 0 | integer | |
| }] | | |
| } | | |
| } | | |

Properties

deplete

Note that in this case the values in this structure report the amount and number of each denomination that have actually been moved during the depletion command.

deplete/numberOfItemsReceived

Total number of items received in the target cash unit during execution of this command.

deplete/numberOfItemsRejected

Total number of items rejected during execution of this command.

deplete/depleteSourceResults

Array of DepleteSpourceResult structures. In the case where one item type has several releases and these are moved, or where items are moved from a multi denomination cash unit to a multi denomination cash unit, each source can move several *noteID* item types.

For example: If one single source was specified with the *depleteSources* input structure, and this source moved two different *noteID* item types, then the *depleteSourceResults* array will have two elements. Or if two sources were specified and the first source moved two different *noteID* item types and the second source moved three different *noteID* item types, then the *depleteSourceResults* array will have five elements.

deplete/depleteSourceResults/cashunitSource

Object name of the cash unit (as stated by the [CashManagement.GetCashUnitInfo](#) command) from which items have been removed.

deplete/depleteSourceResults/noteID

Identification of item type. The note ID represents the item identifiers reported by the [CashAcceptor.BanknoteTypes](#) command.

deplete/depleteSourceResults/numberOfItemsRemoved

Total number of items removed from this source cash unit of the *noteID* item type. A zero value will be returned if this source cash unit did not move any items of this item type, for example due to a cash unit or transport jam.

6.4 - Unsolicited Messages

6.4.1 - [CashAcceptor.ItemsTakenEvent](#)

This service specifies that items presented to the user have been taken. This event may be generated at any time.

| Payload | Type | Required |
|----------------------|--------|----------|
| { | | |
| "position": "inLeft" | string | |
| } | | |

Properties

position

Specifies the position where the items have been inserted. Following values are possible:

"inLeft": Items taken from the left input position.

"inRight": Items taken from the right input position.

"inCenter": Items taken from the center input position.

"inTop": Items taken from the top input position.

"inBottom": Items taken from the bottom input position.

"inFront": Items taken from the front input position.

"inRear": Items taken from the rear input position.

"outLeft": Items taken from the left output position.

"outRight": Items taken from the right output position.

"outCenter": Items taken from the center output position.

"outTop": Items taken from the top output position.

"outBottom": Items taken from the bottom output position.

"outFront": Items taken from the front output position.

"outRear": Items taken from the rear output position.

6.4.2 - CashAcceptor.ItemsPresentedEvent

This service event specifies that items have been presented to the output position, and the shutter has been opened to allow the user to take the items.

| Payload | Type | Required |
|------------------------------|--------|----------|
| { | | |
| "position": "inLeft", | string | |
| "additionalBunches": "none", | string | |

```
"bunchesRemaining": 0      integer  
}
```

Properties

position

Specifies the position where the items have been inserted. Following values are possible:

"inLeft": Items presented at the left input position.

"inRight": Items presented at the right input position.

"inCenter": Items presented at the center input position.

"inTop": Items presented at the top input position.

"inBottom": Items presented at the bottom input position.

"inFront": Items presented at the front input position.

"inRear": Items presented at the rear input position.

"outLeft": Items presented at the left output position.

"outRight": Items presented at the right output position.

"outCenter": Items presented at the center output position.

"outTop": Items presented at the top output position.

"outBottom": Items presented at the bottom output position.

"outFront": Items presented at the front output position.

"outRear": Items presented at the rear output position.

additionalBunches

Specifies whether or not additional bunches of items are remaining to be presented as a result of the current operation. Following values are possible:

"none": No additional bunches remain.

"oneMore": At least one additional bunch remains.

"unknown": It is unknown whether additional bunches remain.

bunchesRemaining

If *additionalBunches* is "oneMore", specifies the number of additional bunches of items remaining to be presented as a result of the current operation. If the number of additional bunches is at least one, but the precise number is unknown, *bunchesRemaining* will be 255 (TODO: Check if there is a better way to represent this state). For any other value of *additionalBunches*, *bunchesRemaining* will be zero.

6.4.3 - CashAcceptor.ItemsInsertedEvent

This service event specifies that items have been inserted into the cash-in position by the user. This event may be generated at any time.

| Payload | Type | Required |
|--|------|----------|
| { "position": "inLeft" string } | | |

Properties

position

Specifies the position where the items have been inserted. Following values are possible:

"inLeft": Items detected in the left input position.

"inRight": Items detected in the right input position.

"inCenter": Items detected in the center input position.

"inTop": Items detected in the top input position.

"inBottom": Items detected in the bottom input position.

"inFront": Items detected in the front input position.

"inRear": Items detected in the rear input position.

"outLeft": Items detected in the left output position.

"outRight": Items detected in the right output position.

"outCenter": Items detected in the center output position.

"outTop": Items detected in the top output position.

"outBottom": Items detected in the bottom output position.

"outFront": Items detected in the front output position.

"outRear": Items detected in the rear output position.

6.4.4 - CashAcceptor.MediaDetectedEvent

This service event is generated if media is detected during a CashAcceptor.Reset command. The parameter on the event specifies the position of the media on completion of the reset. If the device has been unable to successfully move the items found then this parameter will be omitted.

| Payload | Type | Required |
|----------------------------------|---------|----------|
| { | | |
| "cashunit": Add example to YAML, | string | |
| "retractArea": { | object | |
| "outputPosition": "null", | string | |
| "retractArea": "retract", | string | |
| "index": 0 | integer | |
| }, | | |
| "outputPosition": "null" | string | |
| } | | |

Properties

cashunit

If defined, this value specifies the object name (as stated by the [CashManagement.GetCashUnitInfo](#) command) of the single cash unit to be used for the storage of any items found.

If items are to be moved to an output position, this value must be omitted, [retractArea](#) must be omitted and [outputPosition](#) specifies where items are to be moved to. If this value is

omitted and items are to be moved to internal areas of the device, *retractArea* specifies where items are to be moved to or stored.

retractArea

This field is used if items are to be moved to internal areas of the device, including cash units, the intermediate stacker or the transport. The field is only relevant if [cashunit](#) is not defined.

retractArea/outputPosition

Specifies the output position from which to retract the bills. Following values are possible:

"null": The default configuration information should be used. This value is also used to retract items from internal device locations.

"left": Retract items from the left output position.

"right": Retract items from the right output position.

"center": Retract items from the center output position.

"top": Retract items from the top output position.

"bottom": Retract items from the bottom output position.

"front": Retract items from the front output position.

"rear": Retract items from the rear output position.

retractArea/retractArea

This value specifies the area to which the items are to be retracted. Following values are possible:

"retract": Retract the items to a retract cash unit.

"reject": Retract the items to a reject cash unit.

"transport": Retract the items to the transport.

"stacker": Retract the items to the intermediate stacker area.

"billCassettes": Retract the items to item cassettes, i.e. cash-in and recycle cash units.

"cashIn": Retract the items to a cash-in cash unit. The *itemType* of the cash-in cash unit defined in [CashManagement.CashUnitInfo](#) must include "all" and "unfit".

retractArea/index

If *retractArea* is set to "retract" this field defines the position inside the retract cash units into which the cash is to be retracted. *index* starts with a value of one (1) for the first retract position and increments by one for each subsequent position. The maximum value of *index* is the sum of the *maximum* of each retract cash unit.

If *retractArea* is set to "cashIn" this field defines the cash unit under the "cashIn" cash units into which the cash is to be retracted. *index* corresponds to the cash unit *number* defined in CashManagement.CashUnitInfo.

If *retractArea* is not set to "retract" or "cashIn" then the value of this field is ignored.

outputPosition

The output position to which items are to be moved. This field is only used if *number* is zero and *nretractArea* is omitted. Following values are possible:

"null": Take the default configuration.

"left": Move items to the left output position.

"right": Move items to the right output position.

"center": Move items to the center output position.

"top": Move items to the top output position.

"bottom": Move items to the bottom output position.

"front": Move items to the front output position.

"rear": Move items to the rear output position.

6.4.5 - CashAcceptor.ShutterStatusChangedEvent

Within the limitations of the hardware sensors this service event is generated whenever the status of a shutter changes. The shutter status can change because of an explicit, implicit or manual operation depending on how the shutter is operated.

| Payload | Type | Required |
|-----------------------|--------|----------|
| { | | |
| "position": "inLeft", | string | |

```
"shutter": "closed"    string  
}
```

Properties

position

Specifies one of the input or output positions whose shutter status has changed. Following values are possible:

- "inLeft": Left input position.
- "inRight": Right input position.
- "inCenter": Center input position.
- "inTop": Top input position.
- "inBottom": Bottom input position.
- "inFront": Front input position.
- "inRear": Rear input position.
- "outLeft": Left output position.
- "outRight": Right output position.
- "outCenter": Center output position.
- "outTop": Top output position.
- "outBottom": Bottom output position.
- "outFront": Front output position.
- "outRear": Rear output position.

shutter

Specifies the new state of the shutter. Following values are possible:

- "closed": The shutter is closed.
- "open": The shutter is opened.
- "jammed": The shutter is jammed.

"unknown": Due to a hardware error or other condition, the state of the shutter cannot be determined.

6.4.6 - CashAcceptor.CountAccuracyChangedEvent

This event is generated when information about the accuracy of *count* of a cash unit is changed.

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "cashUnitCountStatus": { | object | |
| "additionalProperties": { | object | |
| "physicalPositionName": Add example to YAML, | string | |
| "accuracy": "notSupported" | string | |
| } | | |
| } | | |
| } | | |

Properties

cashUnitCountStatus

Object containing cashUnitCountStatus objects. cashUnitCountStatus objects use the same names as used in [CashManagement.GetCashUnitInfo](#).

cashUnitCountStatus/additionalProperties/physicalPositionName

A name identifying the physical location of the cash unit within the CashAcceptor. This field can be used to identify shared cash units/media bins.

cashUnitCountStatus/additionalProperties/accuracy

Describes the accuracy of *count*. Following values are possible:

"notSupported": The hardware is not capable to determine the accuracy of *count*.

"accurate": The *count* is expected to be accurate. The notes were previously counted or

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

replenished and there have since been no events that might have introduced inaccuracy. This value will be reported as a result of the following commands: Replenish and CashUnitCount.

"accurateSet": The *count* is expected to be accurate. The notes were previously set and there have since been no events that might have introduced inaccuracy.

"inaccurate": The *count* is likely to be inaccurate. A jam, picking fault, or some other event may have resulted in a counting inaccuracy.

"unknown": The accuracy of *count* cannot be determined. This may be due to cash unit insertion or some other hardware event.

7 - Key Management Interface

This chapter defines the Key Management interface functionality and messages.

7.1 - Summary

This section describes the general interface for the following functions:

- Loading of encryption keys
- EMV 4.0 PIN blocks, EMV 4.0 public key loading, static and dynamic data verification

Important Notes:

- This revision of this specification does not define all key management procedures; some key management is still vendor-specific.
- Key space management is customer-specific, and is therefore handled by vendor-specific mechanisms.

Key values are passed to the API as binary hexadecimal values, for example:
0123456789ABCDEF = 0x01 0x23 0x45 0x67 0x89 0xAB 0xCD 0xEF. When hex values are passed to the API within strings, the hex digits 0xA to 0xF can be represented by characters in the ranges 'a' to 'f' or 'A' to 'F'. The following commands and events were initially added to support the German ZKA standard, but may also be used for other national standards:

Certain levels of the PCI EPP security standards specify that if a key encryption key is deleted or replaced, then all keys in the hierarchy under that key encryption key are also removed. Key encryption keys have the keyEncKey type of access. Clients can check impact of key deletion using Keydetail.

7.2 - General Information

7.2.1 - RKL Terminology

This section provides extended explanation of concepts and functionality needing further clarification. The terminology as described below is used within the following sections.

Definitions and
Abbreviations

| | |
|-----|--|
| ATM | Automated Teller Machine, used here for any type of self-service terminal, regardless whether it actually dispenses cash |
|-----|--|

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

| | |
|------------------|--|
| CA | Certificate Authority |
| Certificate | A data structure that contains a public key and a name that allows certification of a public key belonging to a specific individual. This is certified using digital signatures. |
| HOST | The remote system that an ATM communicates with. |
| KTK | Key Transport Key |
| PKI | Public Key Infrastructure |
| Private Key | That key of an entity's key pair that should only be used by that entity. |
| Public Key | That key of an entity's key pair that can be made public. |
| Symmetric Key | A key used with symmetric cryptography |
| verification Key | A key that is used to verify the validity of a certificate |
| signatureIssuer | An entity that signs the ATM's public key at production time, may be the ATM manufacturer |

Notation of Cryptographic Items and Functions

| | |
|----------|--|
| SKE | The private key belonging to entity E |
| PKE | The public key belonging to entity E |
| SKATM | The private key belonging to the ATM/PIN |
| PKATM | The public key belonging to the ATM/PIN |
| SKHOST | The private key belonging to the Host |
| PKHOST | The public key belonging to the Host |
| SKSI | The private key belonging to Signature Issuer |
| PKSI | The public key belonging to Signature Issuer |
| SKROOT | The root private key belonging to the Host |
| PKROOT | The root public key belonging to the Host |
| KNAME | A symmetric key |
| CertHOST | A Certificate that contains the public verification of the host and is signed by a trusted Certificate Authority. |
| CertATM | A Certificate that contains the ATM/PIN public verification or encipherment key, which is signed by a trusted Certificate Authority. |
| CertCA | The Certificate of a new Certificate Authority |
| RATM | Random Number of the ATM/PIN |
| IHOST | Identifier of the Host |
| KTK | Key Transport Key |

| | |
|------------------|---|
| RHOST | Random number of the Host |
| IATM | Identifier of the ATM/PIN |
| TPATM | Thumb Print of the ATM/PIN |
| Sign(SKE)[D] | The signing of data block D, using the private key SKE |
| Recover(PKE)[S] | The recovery of the data block D from the signature S, using the private key PKE |
| RSACrypt(PKE)[D] | RSA Encryption of the data block D using the public key PKE |
| Hash [M] | Hashing of a message M of arbitrary length to a 20 Byte hash value |
| Des(K) [D] | DES encipherment of an 8 byte data block D using the secret key K |
| Des-1(K) [D] | DES decipherment of an 8 byte data block D using the 8 byte secret key K |
| Des3(K) [D] | Triple DES encipherment of an 8 byte data block D using the 16 byte secret key K = (KL) |
| Des3-1 (K) [D] | Triple DES decipherment of an 8 byte data block D using the 16 byte secret key K = (KL) |
| RndE | A random number created by entity E |
| UIE | Unique Identifier for entity E |
| (A B) | Concatenation of A and B |

7.2.2 - Remote Key Loading Using Signatures

RSA Data Authentication and Digital Signatures

Digital signatures rely on a public key infrastructure (PKI). The PKI model involves an entity, such as a Host, having a pair of encryption keys – one private, one public. These keys work in consort to encrypt, decrypt and authenticate data. One way authentication occurs is through the client of a digital signature. For example:

1. The Host creates some data that it would like to digitally sign;
2. Host runs the data through a hashing algorithm to produce a hash or digest of the data. The digest is unique to every block of data – a digital fingerprint of the data, much smaller and therefore more economical to encrypt than the data itself.
3. Digest is encrypted with the Host's private key.

This is the digital signature – a data block digest encrypted with the private key. The Host then sends the following to the ATM:

1. Data block.

2. Digital signature.
3. Host's public key.

To validate the signature, the ATM performs the following:

1. ATM runs data through the standard hashing algorithm – the same one used by the Host – to produce a digest of the data received. Consider this digest2;
2. ATM uses the Host's public key to decrypt the digital signature. The digital signature was produced using the Host's private key to encrypt the data digest; therefore, when decrypted with the Host's public key it produces the same digest. Consider this digest1. Incidentally, no other public key in the world would work to decrypt digest1 – only the public key corresponding to the signing private key.
3. ATM compares digest1 with digest2.

If digest1 matches digest2 exactly, the ATM has confirmed the following:

- Data was not tampered with in transit. Changing a single bit in the data sent from the Host to the ATM would cause digest2 to be different than digest1. Every data block has a unique digest; therefore, an altered data block is detected by the ATM.
- Public key used to decrypt the digital signature corresponds to the private key used to create it. No other public key could possibly work to decrypt the digital signature, so the ATM was not handed someone else's public key. This gives an overview of how Digital Signatures can be used in Data Authentication. In particular, Signatures can be used to validate and securely install Encryption Keys. The following section describes Key Exchange and the use of Digital signatures.

RSA Secure Key Exchange using Digital Signatures

In summary, both end points, the ATM and the Host, inform each other of their Public Keys. This information is then used to securely send the PIN device Master Key to the ATM. A trusted third party, the Signature Issuer, is used to generate the signatures for the Public keys of each end point, ensuring their validity.

The detail of this is as follows:

Purpose: The Host wishes to install a new master key (KM) on the ATM securely.

Assumptions:

1. The Host has obtained the Public Key (PK SI) from the Signature Issuer.
2. The Host has provided the Signature Issuer with its Public Key (PK HOST), and receives the corresponding signature Sign(SK SI)[PK HOST]. The Signature Issuer uses its own Private Key (SK SI) to create this signature.
3. In the case where Enhanced Remote Key Loading is used, the host has provided the Signature Issuer with its Public Key (PK ROOT), and receives the corresponding

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

signature Sign(SK SI)[PK ROOT]. The host has generated another key pair PKHOST and SKHOST and signs the PKHOST with the SKROOT.

4. (Optional) The host obtains a list of the valid encryption module's Unique Identifiers. The Signature Issuer installs a Signature Sign(SK SI)[UI ATM] for the Unique Id (UI ATM) on the ATM encryption module. The Signature Issuer uses SKSI to do this.
5. The Signature Issuer installs its Public Key (PK SI) on the ATM encryption module. It also derives and installs the Signature Sign(SK SI)[PK ATM] of the ATM encryption module's Public Key (PK ATM) on the ATM encryption module. The Signature Issuer uses SKSI to do this.
6. The ATM encryption module additionally contains its own Public (PK ATM) and Private Key (SK ATM).

Step 1

The ATM KeyManagement sends its Public Key to the Host in a secure structure:

The ATM KeyManagement sends its ATM Public Key with its associated Signature. When the Host receives this information it will use the Signature Issuer's Public Key to validate the signature and obtain the ATM Public Key.

The command used to export the encryption module's public key securely as described above is [KeyManagement.ExportRsaIssuerSignedItem..](#)

Step 2 (Optional)

The Host verifies that the key it has just received is from a valid sender.

It does this by obtaining the encryption module unique identifier. The ATM KeyManagement sends its Unique Identifier with its associated Signature. When the Host receives this information it will use the Signature Issuer's Public Key to validate the signature and retrieve the Encryption Module Unique Identifier. It can then check this against the list it received from the Signature Issuer.

The command used to export the encryption module Unique Identifier is [KeyManagement.ExportRsaIssuerSignedItem.](#)

Step 3 (Enhanced Remote Key Loading only)

The Host sends its root public key to the ATM KeyManagement:

The Host sends its Root Public Key (PKROOT) and associated Signature. The ATM encryption module verifies the signature using PKSI and stores the key.

The command used to import the host root public key securely as described above is [ImportRsaPublicKey.](#)

Step 4

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

The Host sends its public key to the ATM KeyManagement:

The Host sends its Public Key (PK HOST) and associated Signature. The ATM encryption module verifies the signature using PKSI (or PKROOT in the Enhanced Remote Key Loading Scheme) and stores the key.

The command used to import the host public key securely as described above is ImportRsaPublicKey.

Step 5

The ATM KeyManagement receives its Master Key from the Host:

The Host encrypts the Master Key (KM) with PKATM. A signature for this is then created using SKHOST. The ATM encryption module will then validate the signature using PKHOST and then obtain the master key by decrypting using SKATM.

The commands used to exchange master symmetric keys as described above are:

- [KeyManagement.StartKeyExchange](#)
- [ImportRsaSignedDesKey](#)

Step 6 – Alternative including random number The host requests the ATM KeyManagement to begin the DES key transfer process and generate a random number.

The Host encrypts the Master Key (KM) with PKATM. A signature for the random number and encrypted key is then created using SKHOST.

The ATM encryption module will then validate the signature using PKHOST, verify the random number and then obtain the master key by decrypting using SKATM.

The commands used to exchange master symmetric keys as described above are:

- [KeyManagement.StartKeyExchange](#)
- [ImportRsaSignedDesKey](#)

The following diagrams summaries the key exchange process described above:

[Default Keys and Security Item loaded during manufacture](#)

Several keys and a security item which are mandatory for the 2 party/Signature authentication scheme are installed during manufacture. These items are given fixed names so multi-vendor clients can be developed without the need for vendor specific configuration tools.

| Item Name | Item Type | Signed by | Description |
|-------------------|------------|-----------|---------------------------------|
| “sigIssuerVendor” | Public Key | N/A | The public key of the signature |

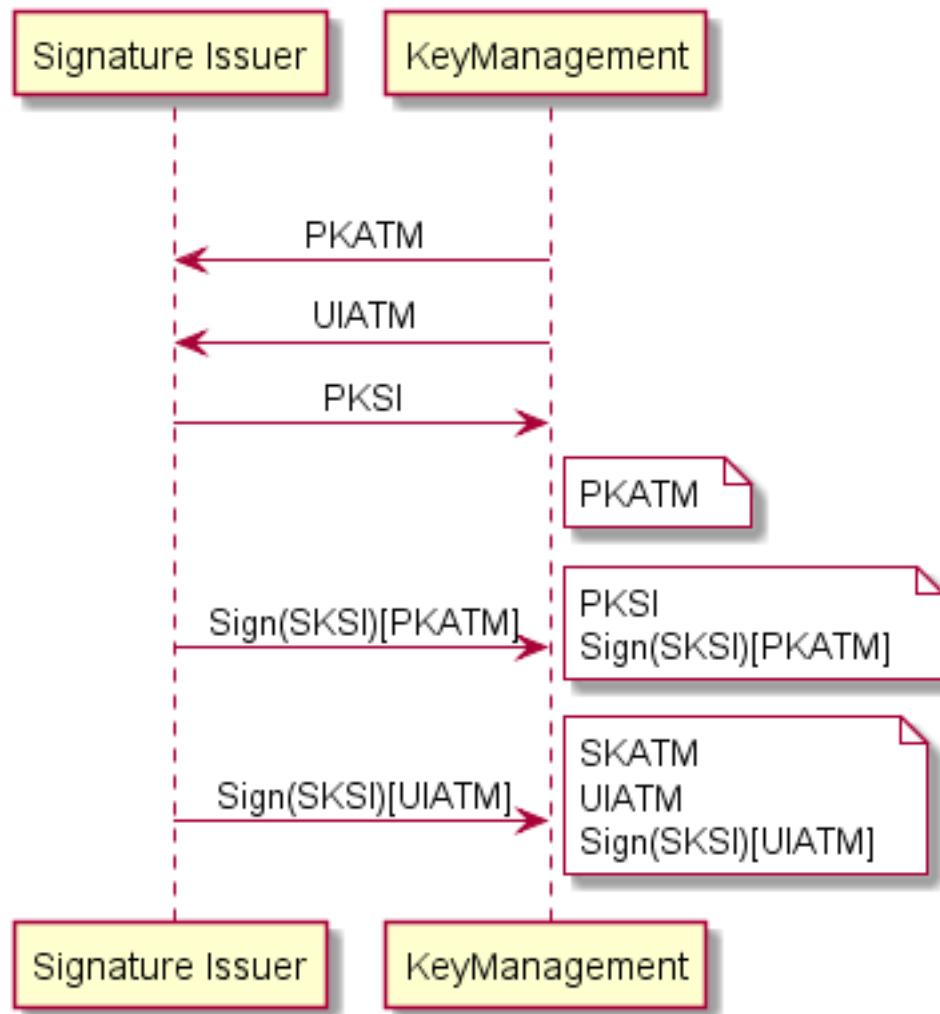
| | | | |
|---------------|-------------------------|---|---|
| “eppCryptKey” | Public/Private key-pair | The private key associated with sigIssuerVendor | issuer, i.e. PKSI The key-pair used to encrypt and encrypt the symmetric key, i.e SK ATM and PK ATM . The public key is used for encryption by the host and the private for decryption by the epp. |
|---------------|-------------------------|---|---|

In addition the following optional keys can be loaded during manufacture.

| Item Name | Item Type | Signed by | Description |
|--------------|-------------------------|---|---|
| “eppSignKey” | Public/Private key-pair | The private key associated with sigIssuerVendor | A key-pair where the private key is used to sign data, e.g. other generated key pairs |

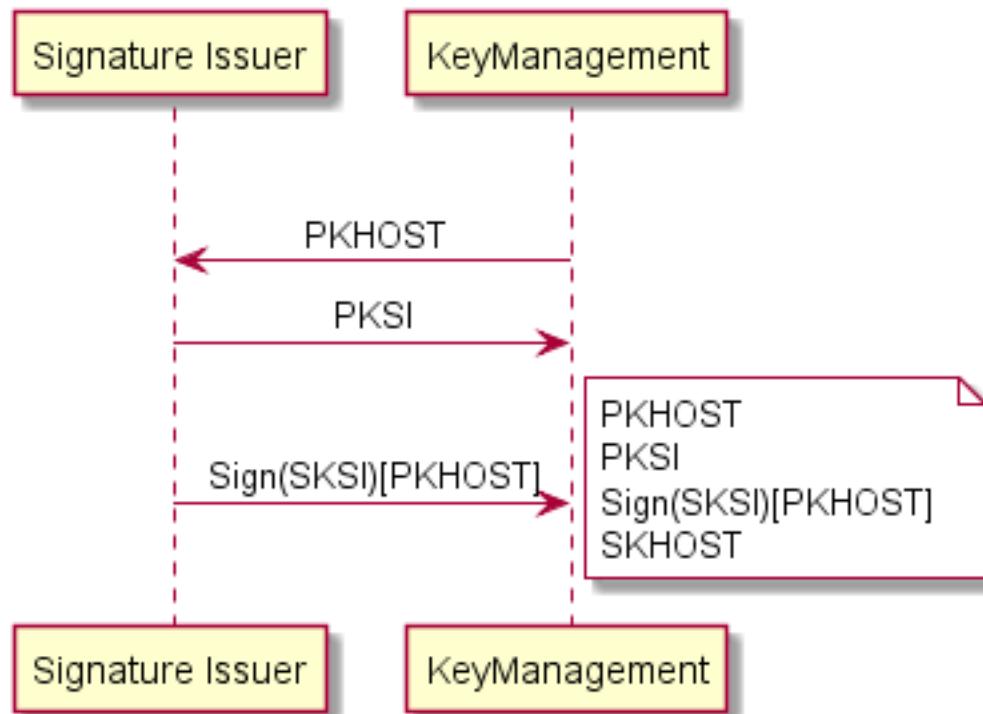
7.2.3 - Initialization Phase – Signature Issuer and ATM PIN

This would typically occur in a secure manufacturing environment.



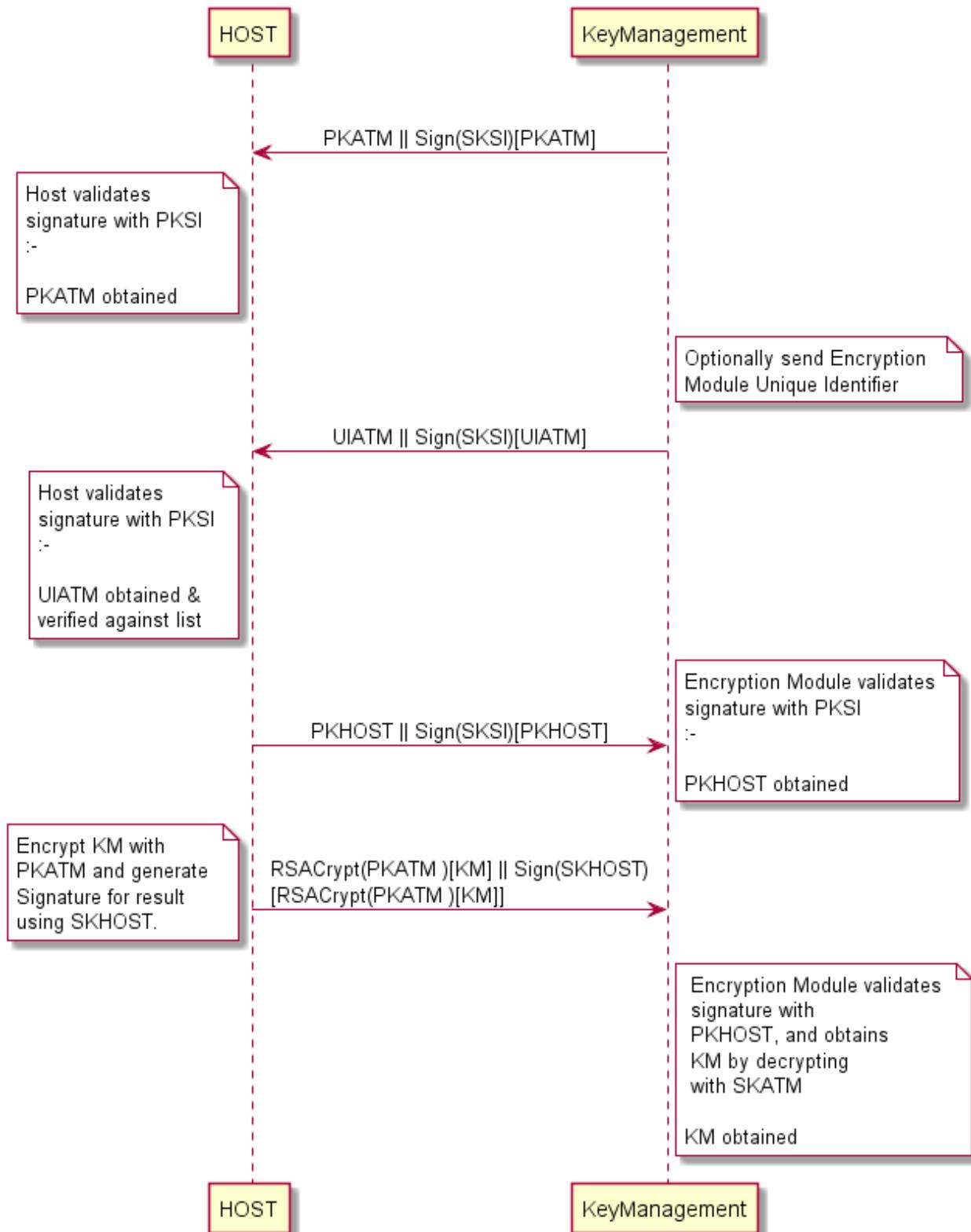
7.2.4 - Initialization Phase – Signature Issuer and Host

This would typically occur in a secure offline environment.



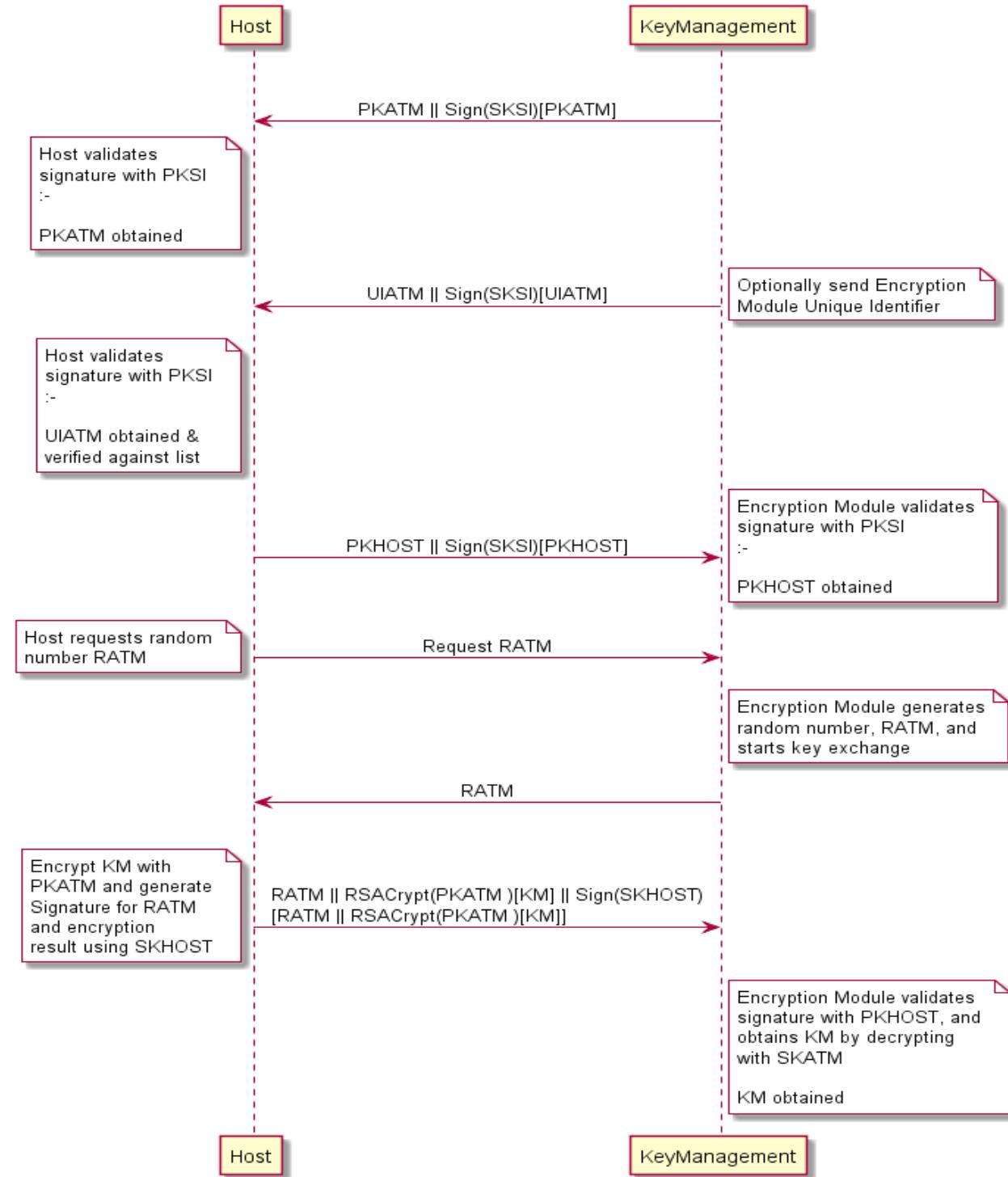
7.2.5 - Key Exchange – Host and ATM PIN

This following is a typical interaction for the exchange of the initial symmetric master key in a typical ATM Network. The following is the recommended sequence of interchanges.



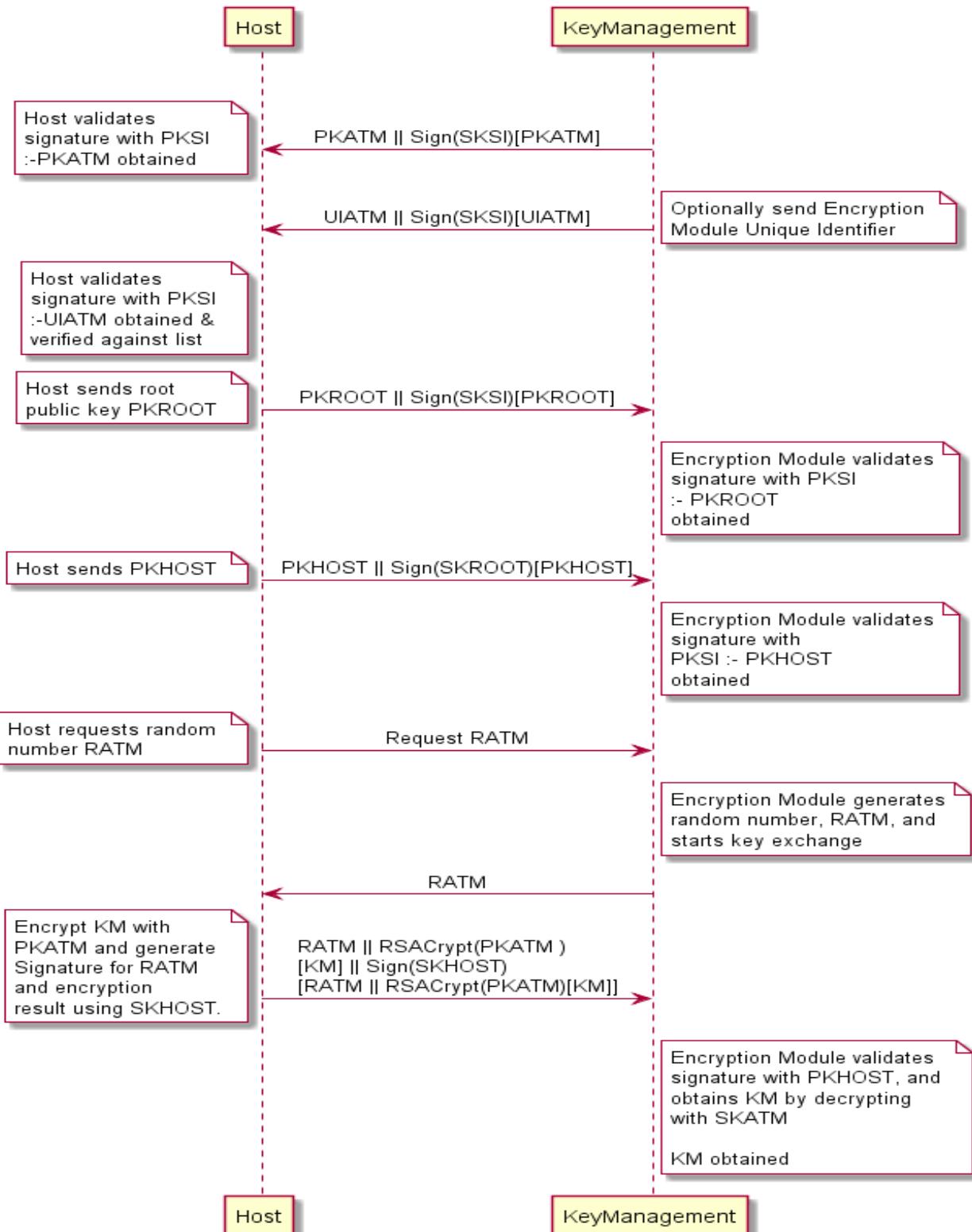
7.2.6 - Key Exchange (with random number) – Host and ATM PIN

This following is a typical interaction for the exchange of the initial symmetric master key when the PIN device Service Provider supports the [KeyManagement.StartKeyExchange](#) command.



7.2.7 - Enhanced RKL, Key Exchange (with random number) – Host and ATM PIN

This following is a typical interaction for the exchange of the initial symmetric master key when the PIN device and Service Provider supports the Enhanced Signature Remote Key Loading scheme.



7.2.8 - Remote Key Loading Using Certificates

The following sections demonstrate the proper usage of the CEN KeyManagement interface to accomplish Remote Key Loading using Certificates. There are sequence diagrams to demonstrate how the CEN KeyManagement interface can be used to complete each of the TR34 operations.

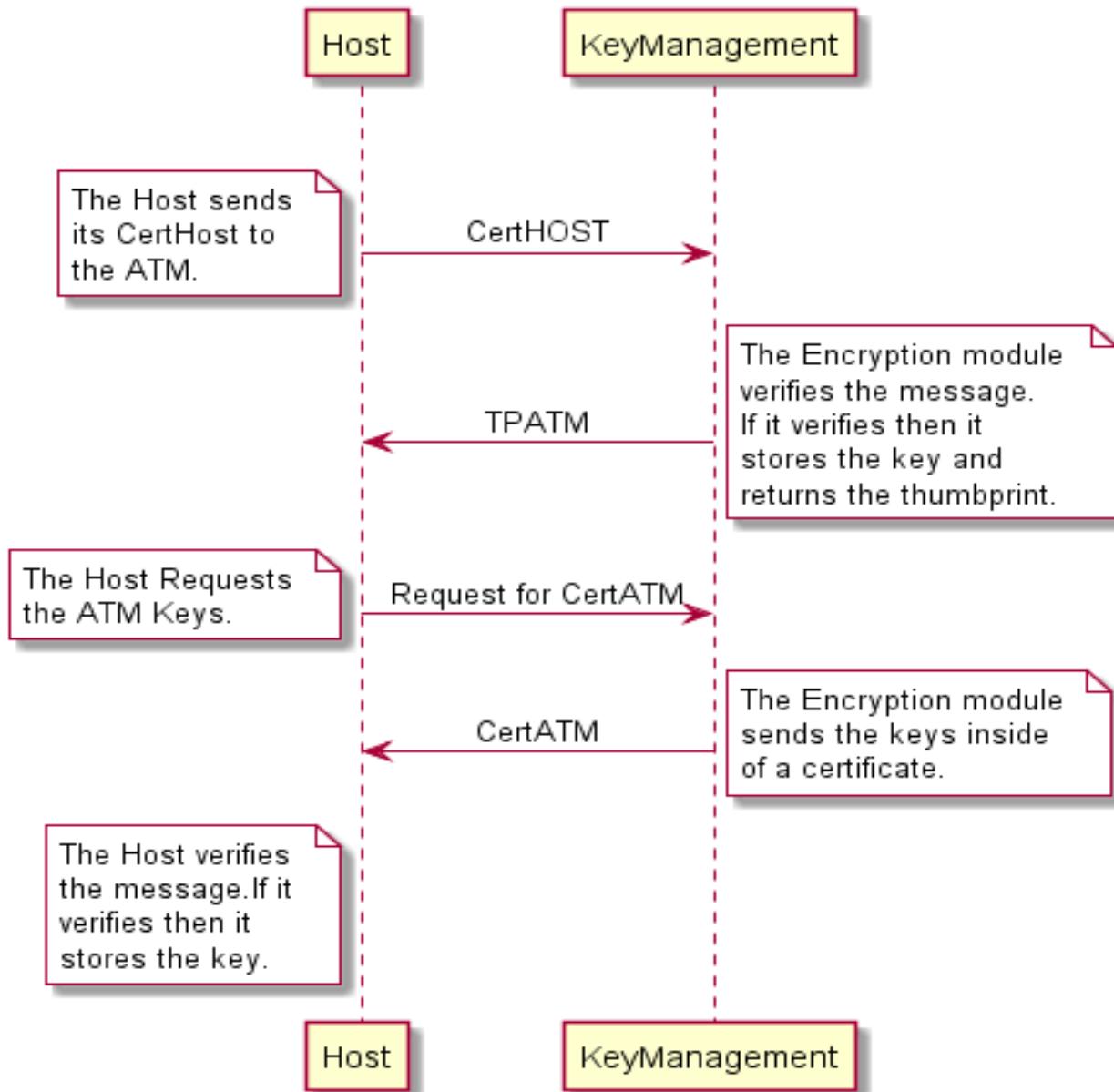
7.2.9 - Certificate Exchange and Authentication

In summary, both end points, the ATM and the Host, inform each other of their Public Keys. This information is then used to securely send the PINS device Master Key to the ATM. A trusted third party, Certificate Authority (or a HOST if it becomes the new CA), is used to generate the certificates for the Public Keys of each end point, ensuring their validity.
NOTE: The [KeyManagement.LoadCertificate](#) and [KeyManagement.GetCertificate](#) do not necessarily need to be called in the order below. This way though is the recommend way.

The following flow is how the exchange authentication takes place:

- [KeyManagement.LoadCertificate](#) is called. In this message contains the host certificate, which has been signed by the trusted CA. The encryptor uses the Public Key of the CA (loaded at the time of production) to verify the validity of the certificate. If the certificate is valid, the encryptor stores the HOST's Public Verification Key.
- Next, [KeyManagement.GetCertificate](#) is called. The encryptor then sends a message that contains a certificate, which is signed by the CA and is sent to the HOST. The HOST uses the Public Key from the CA to verify the certificate. If valid then the HOST stores the encryptor's verification or encryption key (primary or secondary this depends on the state of the encryptor).

The following diagram shows how the Host and ATM Load and Get each other's information to make Remote Key Loading possible:



7.2.10 - Remote Key Exchange

After the above has been completed, the HOST is ready to load the key into the encryptor. The following is done to complete this and the client must complete the Remote Key Exchange in this order:

1. First, the `Keymanagement.StartKeyExchange` is called. This returns RATM from the encryptor to be used in the authenticating the `ImportRSAEnchiperdPKCS7Key` message.

2. Next, ImportRSAEnchiperdPKCS7Key is called. This command sends down the KTK to the encryptor. The following items below show how this is accomplished.

a) HOST has obtained a Key Transport Key and wants to transfer it to the encryptor. HOST constructs a key block containing an identifier of the HOST, IHOST, and the key, KKTK, and enciphers the block, using the encryptor's Public Encryption Key from the [KeyManagement.GetCertificate](#) command.

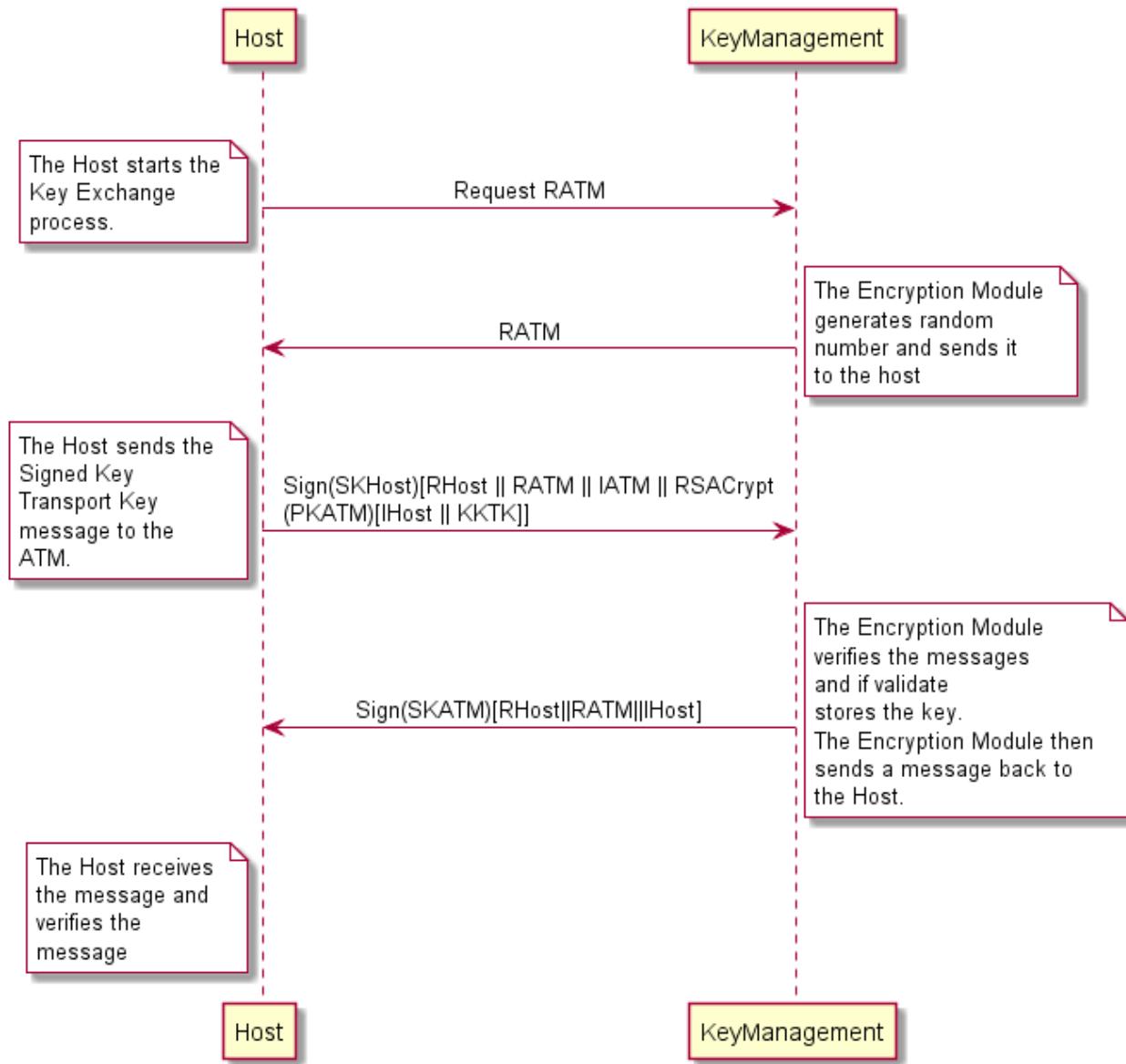
b) After completing the above, the HOST generates random data and builds the outer message containing the random number of the host, RHOST, the random number of the encryptor returned in the [Keymanagement.StartKeyExchange](#) command, RATM, the identifier of the encryptor, IENC, and the enciphered key block. The HOST signs the whole block using its private signature key and sends the message down to the encryptor.

The encryptor then verifies the HOST's signature on the message by using the HOST's Public Verification Key. Then the encryptor checks the identifier and the random number of the encryptor passed in the message to make sure that the encryptor is talking to the right HOST. The encryptor then deciphers the enciphered block using its private verification key. After the message has been deciphered, the encryptor checks the Identifier of the HOST. Finally, if everything checks out to this point the encryptor will load the Key Transport Key. NOTE: If one step of this verification occurs the encryptor will return the proper error to the HOST.

c) After the Key Transport Key has been accepted, the encryptor constructs a message that contains the random number of the host, the random number of the encryptor and the HOST identifier all signed by the private signature key of the encryptor. This message is sent to the host.

d) The HOST verifies the message sent from the encryptor by using the ATM's public verification key. The HOST then checks the identifier of the host and then compares the identifier in the message with the one stored in the HOST. Then checks the random number sent in the message and to the one stored in the HOST. The HOST finally checks the encryptor's random number with the one received in received in the [Keymanagement.StartKeyExchange](#) command.

The following diagram below shows how the Host and ATM transmit the Key Transport Key.

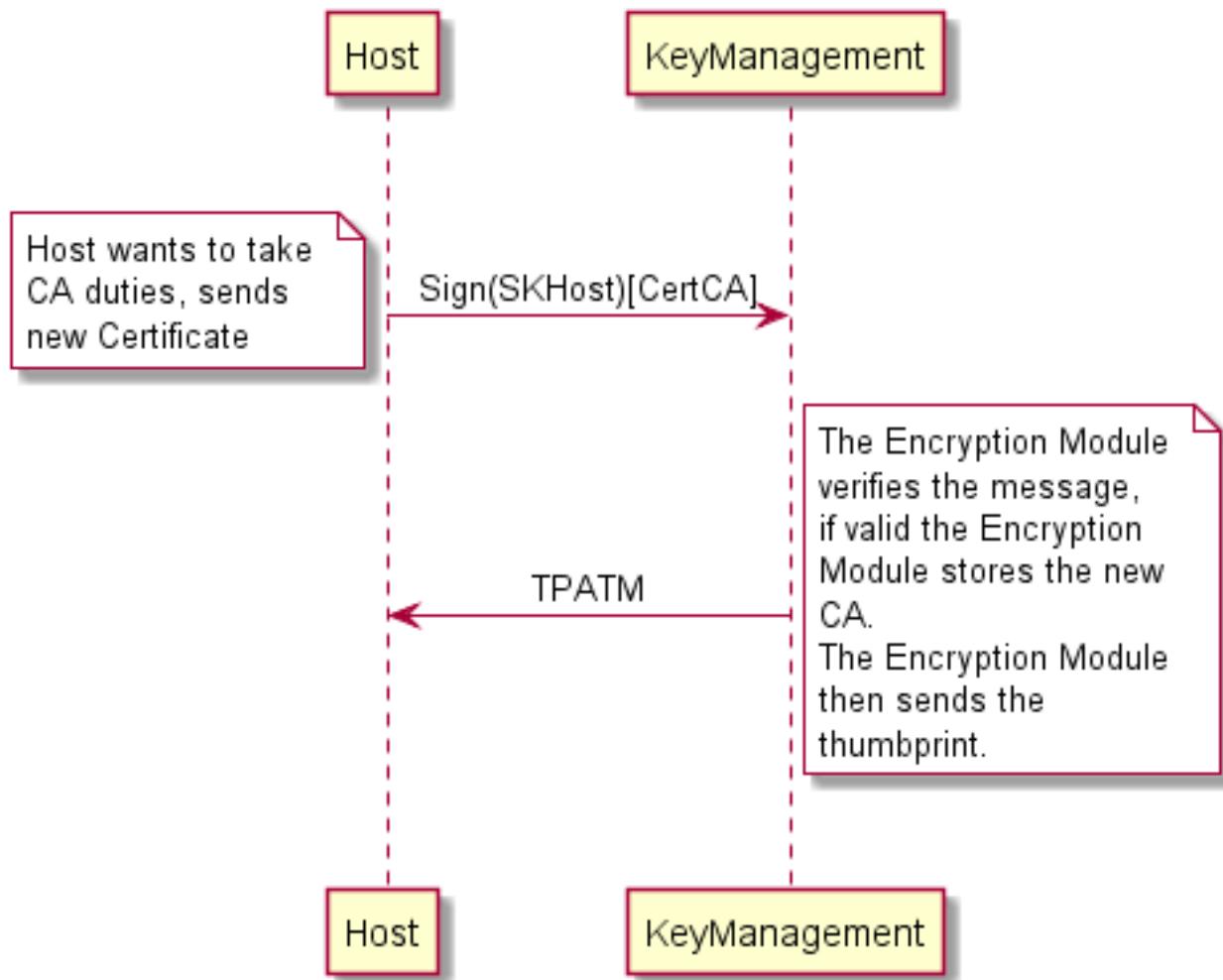


7.2.11 - Replace Certificate

After the key is been loaded into the encryptor, the following could be completed:

- (Optional) ReplaceCertificate. This is called by entity that would like to take over the job of being the CA. The new CA requests a Certificate from the previous Certificate Authority. The HOST must over-sign the message to take over the role of the CA to ensure that the encryptor accepts the new Certificate Authority. The HOST sends the message to the encryptor. The encryptor uses the HOST's Public Verification Key to verify the HOST's signature. The encryptor uses the previous CA's Public Verification Key to verify the signature on the new Certificate sent down in the message. If valid, the EPP stores the new CA's certificate and uses the new CA's

Public Verification Key as its new CA verification key. The diagram below shows how the Host and the ATM communicate to load the new CA.



7.2.12 - Primary and Secondary Certificate

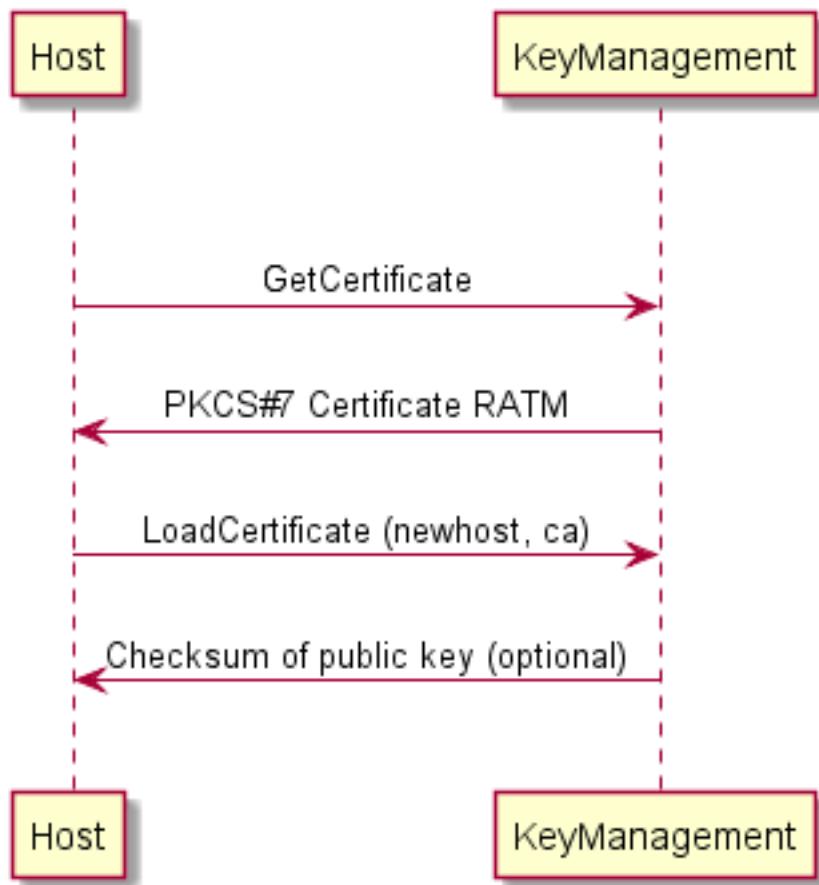
Primary and Secondary Certificates for both the Public Verification Key and Public Encipherment Key are pre-loaded into the encryptor. Primary Certificates will be used until told otherwise by the host via the [KeyManagement.LoadCertificate](#) or [KeyManagement.ReplaceCertificate](#) commands. This change in state will be specified in the pkcs #7 message of the LoadCertificate or [KeyManagement.ReplaceCertificate](#keymanagement.replacecertificate commands. The reason why the host would want to change states is because the host thinks that the Primary Certificates have been compromised.

After the host tells the encryptor to shift to the secondary certificate state, only Secondary Certificates can be used. The encryptor will no longer be able to go back to the Primary State and any attempts from the host to get or load a Primary Certificate will return an

error. When either Primary or Secondary certificates are compromised it is up to the vendor on how the encryptor should be handled with the manufacturer.

7.2.13 - TR34 BIND To Host

This section defines the command to use when transferring a TR34 BIND token. This step is a pre-requisite for all other TR34 operations. The PIN device must be bound to a host before any other TR34 operation will succeed. It is recommended that the encryption certificate retrieved during this process is stored for future use otherwise it will need to be requested prior to every operation.

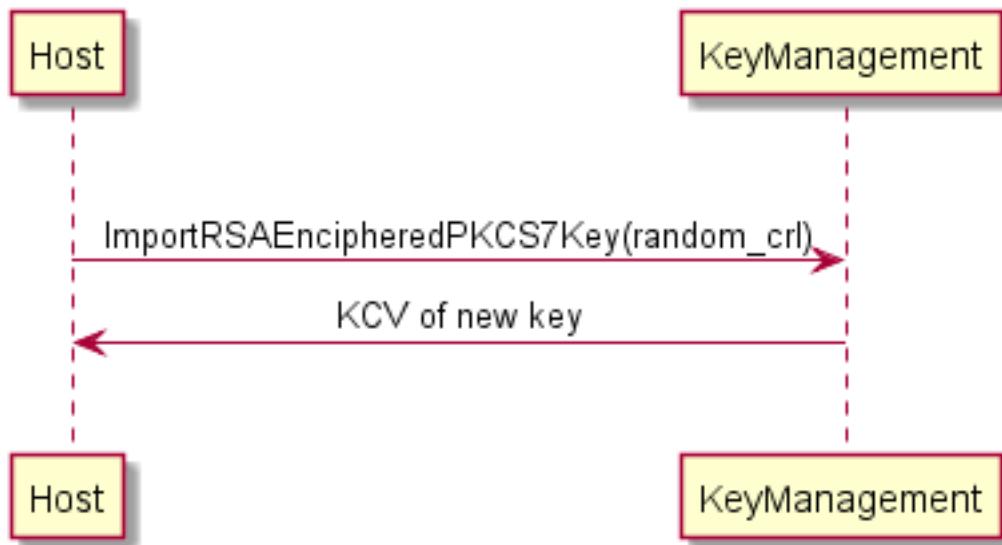


7.2.14 - TR34 Key Transport

There are two mechanisms that can be used to transport symmetric keys under TR34; these are the One Pass and Two Pass protocols. The use of CEN commands for these two protocols are shown in the following sections. NOTE: Refer to CRKLLoadOptions in the Capabilities output structure for an indication of whether the PIN device supports one-pass and/or two-pass protocols.

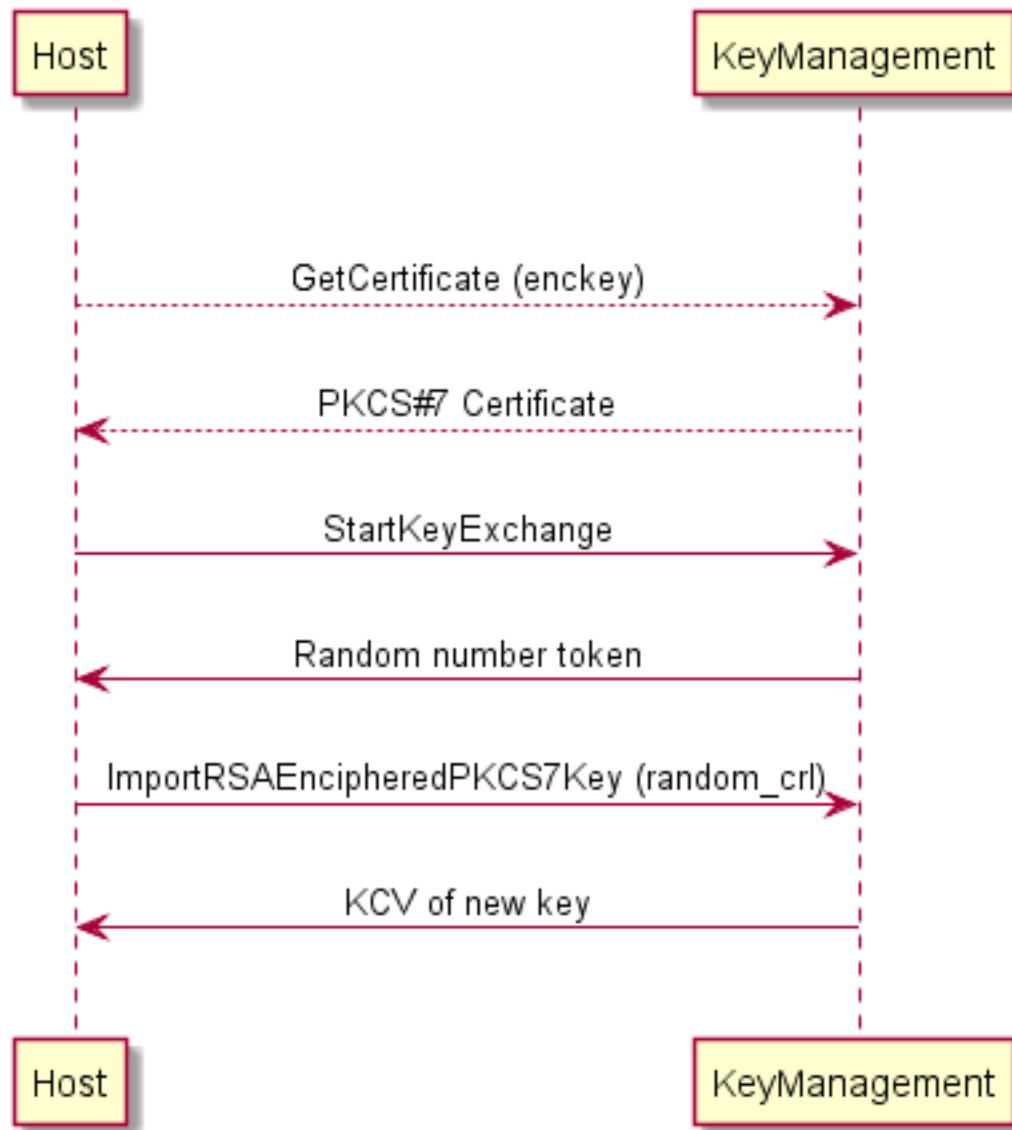
One Pass

This section defines the command to use when transferring a TR34 KEY token (1-pass).
Pre-condition: A successful BIND command has completed such that the KeyManagement is bound to the host.



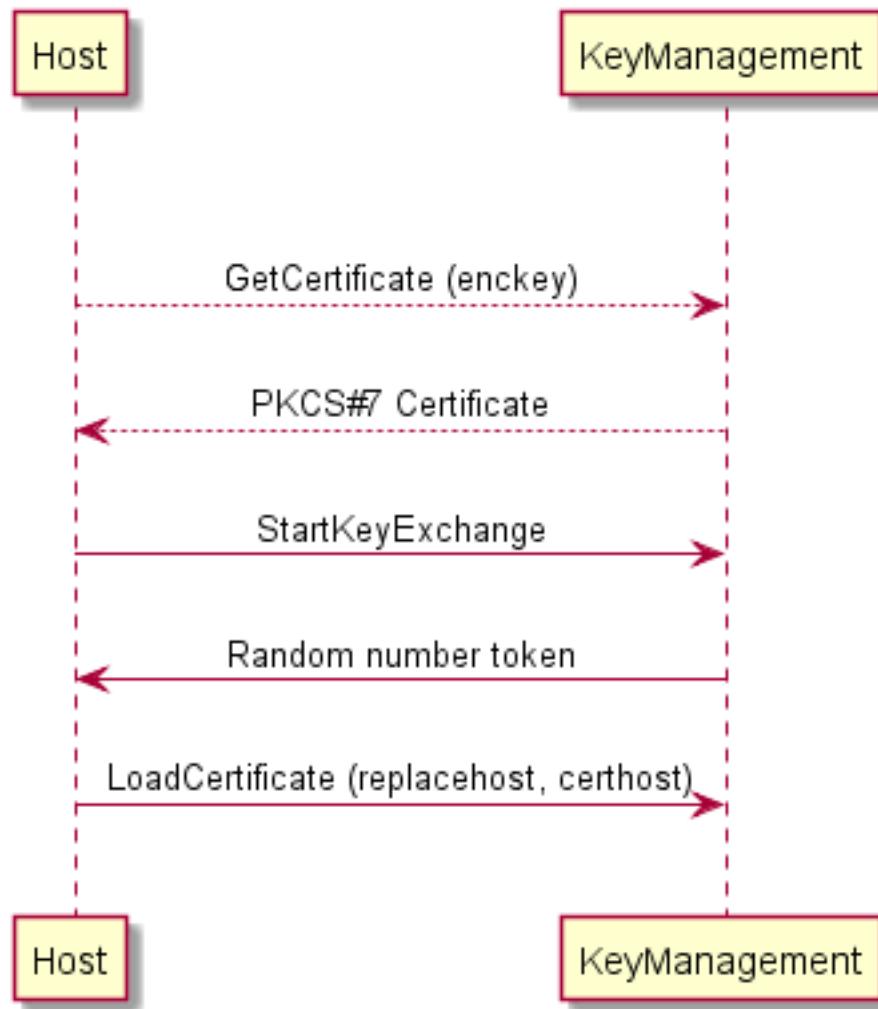
Two Pass

This section defines the command to use when transferring a TR34 KEY token (2-pass).
Pre-condition: A successful BIND command has completed such that the KeyManagement is bound to the host.



7.2.15 - TR34 REBIND To New Host

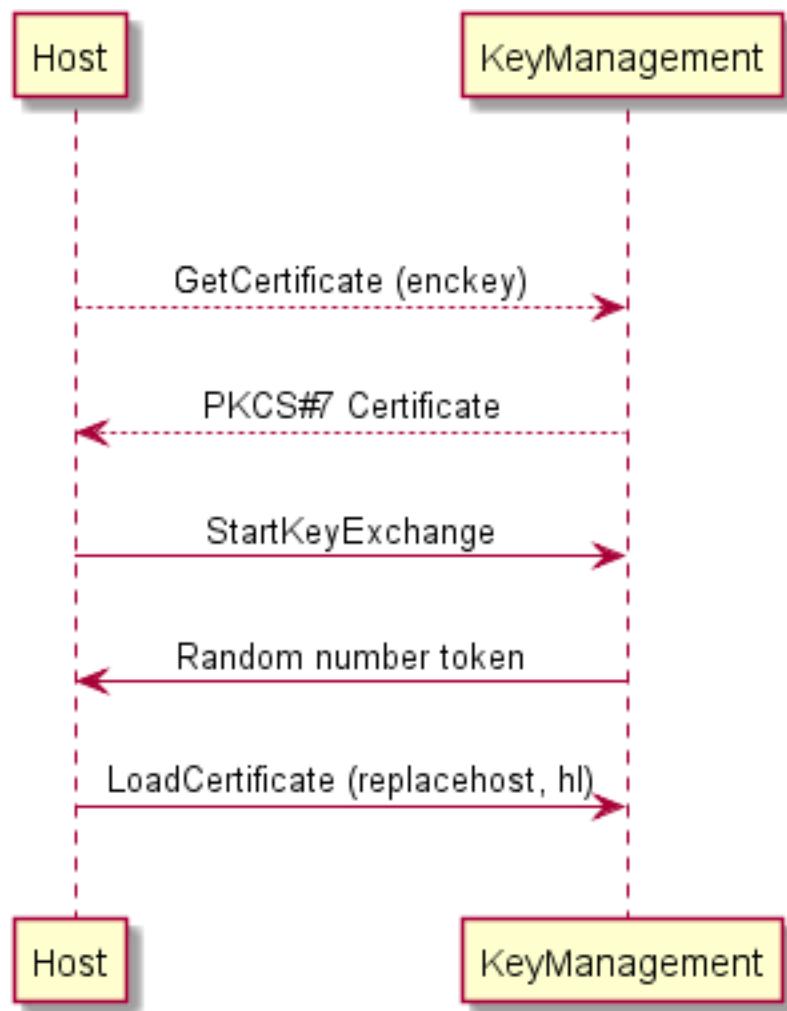
This section defines the command to use when transferring a TR34 REBIND token. Pre-condition: A successful BIND command has completed such that the KeyManagement is bound to the host.



NB: Dotted lines represent commands that are only required if the KeyManagement encryption certificate has not been previously stored by the host.

7.2.16 - TR34 Force REBIND To New Host

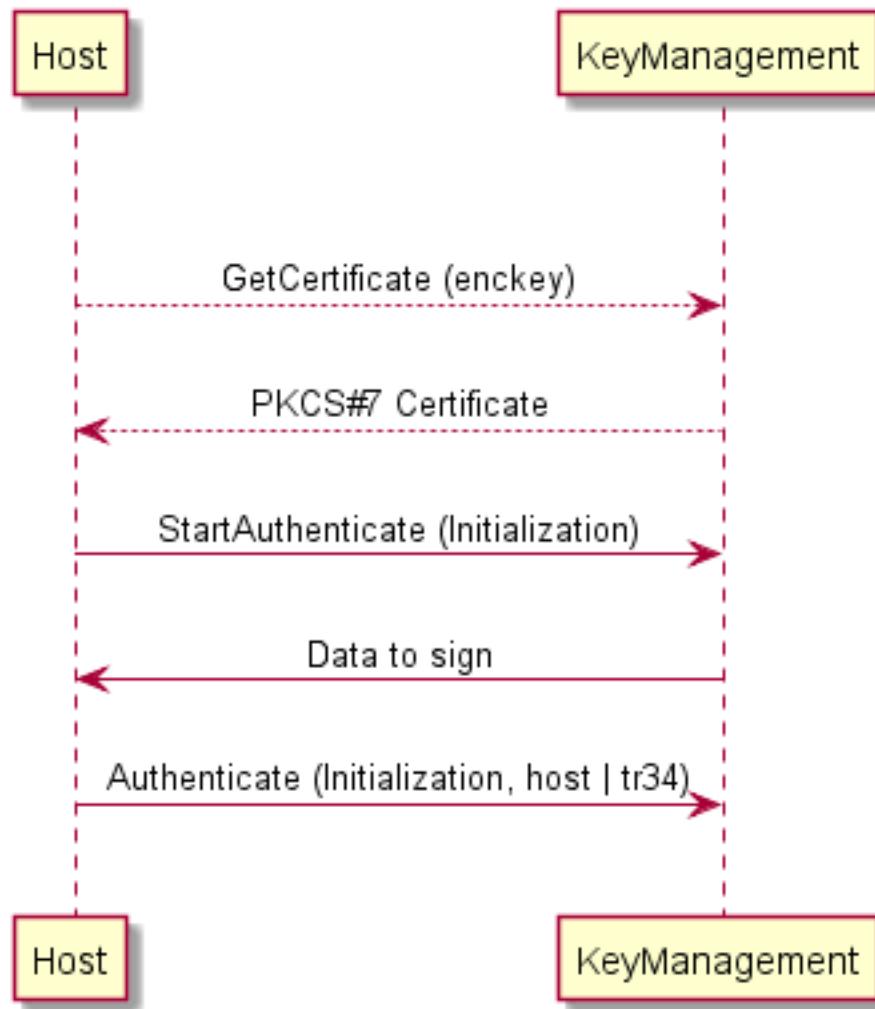
This section defines the command to use when transferring a TR34 Force REBIND token.
 Pre-condition: A successful BIND command has completed such that the KeyManagement is bound to the host.



NB: Dotted lines represent commands that are only required if the KeyManagement encryption certificate has not been previously stored by the host. Although the random number token is requested as part of this operation, it is discarded by the host and is not actually used in the Force Rebind token.

7.2.17 - TR34 UNBIND From Host

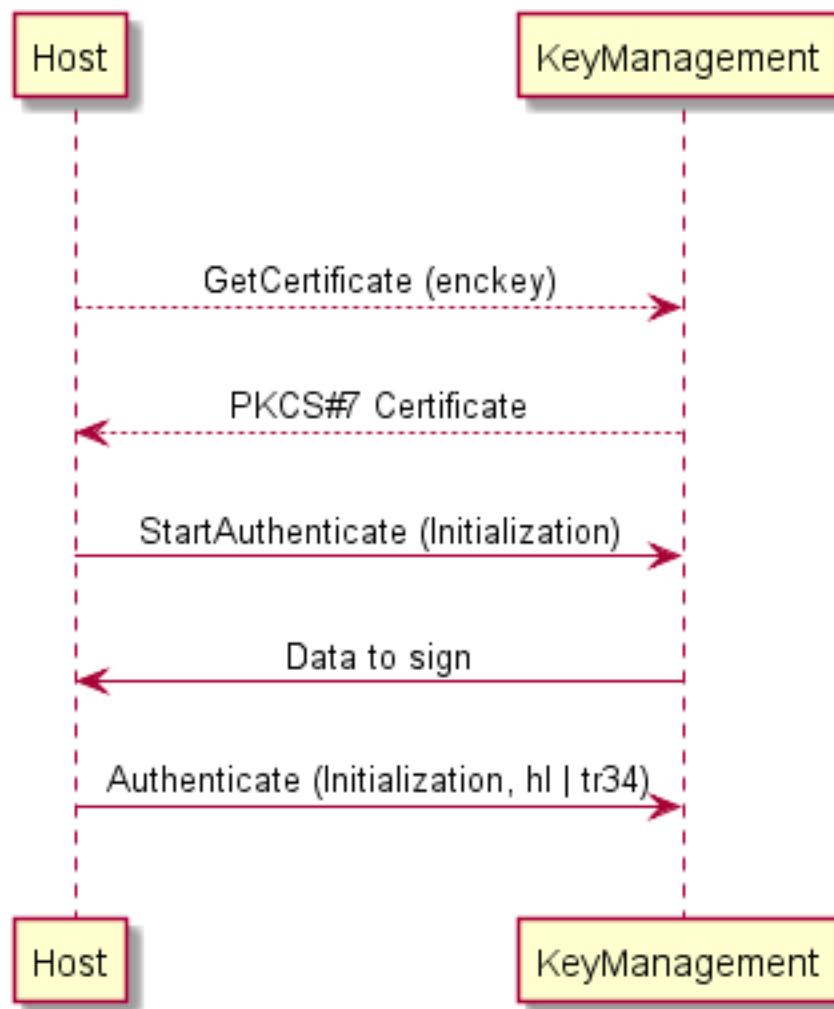
This section defines the command to use when transferring a TR34 UNBIND token. Pre-condition: A successful BIND command has completed such that the KeyManagement is bound to the host.



NB: Dotted lines represent commands that are only required if the KeyManagement encryption certificate has not been previously stored by the host

7.2.18 - TR34 Force UNBIND From Host

This section defines the command to use when transferring a TR34 Force UNBIND token.
 Pre-condition: A successful BIND command has completed such that the KeyManagement is bound to the host.



NB: Dotted lines represent commands that are only required if the KeyManagement encryption certificate has not been previously stored by the host. Although the random number token is requested as part of this operation, it is discarded by the host and is not actually used in the Force Unbind token.

7.2.19 - EMV Support

EMV support by this specification consists in the ability of importing Certification Authority and Chip Card Public Keys, creating the PIN blocks for offline PIN verification and verifying static and dynamic data. This section is used to further explain concepts and functionality that needs further clarification.

The PIN service is able to manage the EMV chip card regarding the card authentication and the RSA local PIN verification. Two steps are mandatory in order to reach these two functions: The loading of the keys which come from the Certification Authorities or from the card itself, and the EMV PIN block management.

The Service Provider is responsible for all key validation during the import process. The client is responsible for management of the key lifetime and expiry after the key is successfully imported

Keys loading

The final goal of a client is to retrieve the keys located on card to perform the operations of authentication or local PIN check (RSA encrypted). These keys are provided by the card using EMV certificates and can be retrieved using a Public Key provided by a Certification Authority. The client should first load the keys issued by the Certification Authority. At transaction time the client will use these keys to load the keys that the client has retrieved from the chip card.

These keys are provided in the following formats:

- Plain text.
- Plain Text with EMV 2000 Verification Data.
- EPI CA (or self signed) format as specified in the Europay International, EPI CA Module Technical – Interface specification Version 1.4.
- pkcsV15 encrypted (as used by GIECB in France).

The following table corresponds to table 4 of the Europay International, EPI CA Module Technical – Interface specification Version 1.4 and identifies the Europay Public Key (self-certified) and the associated data:

| Field Name | Length | Description | Format |
|--|--------|---|--------|
| ID of Certificate Subject | 5 | RID for Europay | Binary |
| Europay public key Index | 1 | Europay public key Index | Binary |
| Subject public key Algorithm Indicator | 1 | Algorithm to be used with the Europay public key Index, set to 0x01 | Binary |
| Subject public key Length | 1 | Length of the Europay public key Modulus (equal to Nca) | Binary |
| Subject public key Exponent Length | 1 | Length of the Europay public key Exponent | Binary |
| Leftmost Digits of Subject public key | Nca-37 | Nca-37 most significant bytes of the Europay public key Modulus | Binary |
| Subject public key | 37 | 37 least significant bytes of the Europay | Binary |

| | | | |
|--------------------------------|-----|---------------------------------|--------|
| Remainder | | public key Modulus | |
| Subject public key Exponent | 1 | Exponent for Europay public key | Binary |
| Subject public key Certificate | Nca | Output of signature algorithm | Binary |

Table 1

The following table corresponds to table 13 of the Europay International, EPI CA Module Technical – Interface specification Version 1.4 and identifies the Europay Public Key Hash code and associated data.

| Field Name | Length | Description | Format |
|--|--------|---|--------|
| ID of Certificate Subject | 5 | RID for Europay | Binary |
| Europay public key Index | 1 | Europay public key Index | Binary |
| Subject public key Algorithm Indicator | 1 | Algorithm to be used with the Europay public key Index, set to 0x01 | Binary |
| Certification Authority public key Check Sum | 20 | Hash-code for Europay public key | Binary |

Table 2

Table 2 corresponds to table 13 of the Europay International, EPI CA Module Technical – Interface specification Version 1.4.

These keys are provided as EMV certificates which come from the chip card in a multiple layer structure (issuer key first, then the ICC keys). Two kinds of algorithm are used with these certificates in order to retrieve the keys: One for the issuer key and the other for the ICC keys (ICC Public Key and ICC PIN encipherment key). The associated data with these algorithms – The PAN (Primary Account Number) and the SDA (Static Data to be Authenticated) - come also from the chip card.

PIN Block Management

The PIN block management is done through the command GetPinBlock. A new format formEmv has been added to indicate to the PIN service that the PIN block must follow the requirements of the EMVCo, Book2 – Security & Key management Version 4.0 document. The parameter customerData is used in this case to transfer to the PIN service the challenge number coming from the chip card. The final encryption must be done using a RSA Public Key. Please note that the client is responsible to send the PIN block to the chip card inside the right APDU.

SHA-1 Digest

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

The SHA-1 Digest is a hash algorithm used by EMV in validating ICC static and dynamic data item. The SHA-1 Digest is supported through the digest command. The client will pass the data to be hashed to the Service Provider. Once the encryptor completes the SHA-1 hash code, the Service Provider will return the 20-byte hash value back to the client.

7.2.20 - ImportKey command Input-Output Parameters

The tables in this section describe the input/output parameters for various scenarios in which the importKey command is used, compared to input/output parameters for older commands that it supercedes.

[Importing a 3DES 16-byte terminal master key using signature-based remote key loading \(SRKL\):](#)

For this example, the following input data is available:

Name of key to be imported = testKey Name of the key used to decrypt the encrypted key value = eppCryptKey Name of the key used to verify the signature = hostKey Encrypted key value = Signature = Usage of the key to be imported = key encrypting key RSA Encipher Algorithm = rsa es oaep RSA Signature Algorithm = rsa ssa pss

[KeyManagement.ImportKey](#) Input Data

| Parameter Name | Example Value |
|-----------------------|-----------------------------------|
| key | testKey |
| decryptKey | eppCryptKey |
| rsaEnchiperAlgorithm | oaep |
| value | <encrypted key value> |
| use | keyEncKey |
| sigKey | hostKey |
| rsaSignatureAlgorithm | pss |
| signature | <signature generated by the host> |

For this example, the following output data is expected:

Key Check Mode = kcv zero Key Check Value = Key Length = double length key

[KeyManagement.ImportKey](#) Output Data

| Parameter Name | Example Value |
|----------------|-------------------|
| keyLength | double |
| keyCheckMode | zero |
| keyCheckValue | <key check value> |

KeyManagement.ImportKey Input Data

| Parameter Name | Example Value |
|-------------------------------|-----------------------------------|
| key | testKey |
| keyAttributes.keyUsage | 'K0' |
| keyAttributes.algorithm | 'T' |
| keyAttributes.modeOfUse | 'D' |
| keyAttributes.cryptoMethod | 0 |
| value | <encrypted key value> |
| decryptKey | eppCryptKey |
| decryptMethod | rsaesOaep |
| verificationData | <signature generated by the host> |
| verifyKey | hostKey |
| verifyAttributes.keyUsage | 'S0' |
| verifyAttributes.algorithm | 'R' |
| verifyAttributes.modeOfUse | 'V' |
| verifyAttributes.cryptoMethod | rsassaPss |
| vendorAttributes | |

KeyManagement.ImportKey Output Data

| Parameter Name | Example Value |
|-------------------------------|-------------------|
| verifyAttributes.keyUsage | '00' |
| verifyAttributes.algorithm | 'T' |
| verifyAttributes.modeOfUse | 'V' |
| verifyAttributes.cryptoMethod | zero |
| verifyData | <key check value> |
| keyLength | 128 |

Importing a 16-byte DES key for pin encryption with a key check value in the input

For this example, the following input data is available:

Name of key to be imported = testKey
 Name of the key used to decrypt the encrypted key
 value = masterKey
 Encrypted key value = Usage of the key to be imported = pin encryption
 Key Check Mode = kcv Zero Key Check Value =

KeyManagement.ImportKey Input Data

| Parameter Name | Example Value |
|----------------|---------------|
|----------------|---------------|

| | |
|-------------------------------|--|
| key | testKey |
| keyAttributes.keyUsage | 'P0' (Similar to use but a more precise key usage) |
| keyAttributes.algorithm | 'T' |
| keyAttributes.modeOfUse | 'E' |
| keyAttributes.cryptoMethod | 0 |
| value | <encrypted key value> |
| decryptKey | masterKey |
| decryptMethod | ecb |
| verificationData | <key check value> |
| verifyKey | |
| verifyAttributes.keyUsage | '00' |
| verifyAttributes.algorithm | 'T' |
| verifyAttributes.modeOfUse | 'V' |
| verifyAttributes.cryptoMethod | zero |
| vendorAttributes | |

Likewise, the following output data is expected:

[KeyManagement.ImportKey](#) Output Data

| Parameter Name | Example Value |
|------------------|---------------|
| verifyAttributes | null |
| verifyData | |
| keyLength | 128 |

[Importing a 16-byte DES key for macing \(MAC Algorithm 3\)](#)

For this example, the following input data is available:

Name of key to be imported = testKey Name of the key used to decrypt the encrypted key
value = masterKey Encrypted key value = Usage of the key to be imported = mac

[KeyManagement.ImportKey](#) Input Data

| Parameter Name | Example Value |
|-------------------------|--|
| key | testKey |
| keyAttributes.keyUsage | 'M3' (Similar to fwUse but a more precise key usage) |
| keyAttributes.algorithm | 'T' |
| keyAttributes.modeOfUse | 'G' |

```

keyAttributes.cryptoMethod 0
value                      <encrypted key value>
decryptKey                 masterKey
decryptMethod               ecb
verificationData
verifyKey
verifyAttributes           null
vendorAttributes

```

[KeyManagement.ImportKey](#) Output Data

| Parameter Name | Example Value |
|-------------------------------|-------------------|
| verifyAttributes.keyUsage | '00' |
| verifyAttributes.algorithm | 'T' |
| verifyAttributes.modeOfUse | 'V' |
| verifyAttributes.cryptoMethod | zero |
| verifyData | <key check value> |
| keyLength | 128 |

[Importing a 2048-bit Host RSA public key](#)

For this example, the following input data is available:

Name of key to be imported = HostKey
 Name of the key used to verify the signature = sigIssuerVendor
 Key value = Signature = Usage of the key to be imported = RSA signature
 verification RSA Signature Algorithm = RSA SSA PSS

[KeyManagement.ImportKey](#) Input Data

| Parameter Name | Example Value |
|-----------------------|--|
| key | hostKey |
| value | <key value> |
| use | rsaPublicVerify |
| sigKey | sigIssuerVendor |
| rsaSignatureAlgorithm | rsassaPss |
| signature | <signature generated by the vendor signature issuer> |

For this example, the following output data is expected:

RSA Key Check Mode = sha256 digest Key Check Value = Key Length = 2048

KeyManagement.ImportKey Output Data

| Parameter Name | Example Value |
|-----------------|-----------------|
| rsaKeyCheckMode | sha256 |
| keyCheckValue | <sha256 digest> |

KeyManagement.ImportKey Input Data

| Parameter Name | Example Value |
|-------------------------------|--|
| key | hostKey |
| keyAttributes.keyUsage | 'S0' |
| keyAttributes.algorithm | 'R' |
| keyAttributes.modeOfUse | 'V' |
| keyAttributes.cryptoMethod | 0 |
| value | <key value> |
| decryptKey | |
| decryptMethod | 0 |
| verificationData | <signature generated by the vendor signature issuer> |
| verifyKey | sigIssuerVendor |
| verifyAttributes.KeyUsage | 'S1' |
| verifyAttributes.Algorithm | 'R' |
| verifyAttributes.ModeOfUse | 'V' |
| verifyAttributes.CryptoMethod | rsassaPss |
| vendorAttributes | |

KeyManagement.ImportKey Output Data

| Parameter Name | Example Value |
|-------------------------------|-----------------|
| verifyAttributes.algorithm | 'R' |
| verifyAttributes.modeOfUse | 'V' |
| verifyAttributes.CryptoMethod | sha256 |
| verifyData | <sha256 digest> |
| keyLength | 2048 |

Importing a 24-byte DES symmetric data encryption key via TR-31 keyblock

For this example, the following input data is available:

Name of key to be imported = testKey
 Name of the key block protection key = masterKey
 Key block =

All rights of exploitation in any form and by any means reserved worldwide for CEN
 national Members.

KeyManagement.ImportKey Input Data

| Parameter Name | Example Value |
|----------------|---------------|
| Key | testKey |
| EncKey | masterKey |
| KeyBlock | <key block> |

For this example, the following output data is expected:

Key Length = triple length (192 bits) DES key

ImportKey Output Data None

KeyManagement.ImportKey Input Data

| Parameter Name | Example Value |
|----------------------------|---------------|
| key | testKey |
| keyAttributes.keyUsage | 'D0' |
| keyAttributes.algorithm | 'T' |
| keyAttributes.modeOfUse | 'E' |
| keyAttributes.cryptoMethod | 0 |
| value | <key block> |
| decryptKey | masterKey |
| decryptMethod | 0 |
| verificationData | |
| verifyKey | |
| verifyAttributes | null |
| vendorAttributes | |

KeyManagement.ImportKey Output Data

| Parameter Name | Example Value |
|------------------|---------------|
| verifyAttributes | null |
| verifyData | |
| keyLength | 192 |

7.2.21 - TR-31 Key Use

This section contains a mapping of key usages as defined for TR-31 to the use values defined in this document. The use values are those defined for the use input/output fields of a number of different KeyManagement commands.

Keys imported within an ANS TR-31 key block have a usage encoded into the key block header (represented by BlockHeader in the KeyDetail commands). This usage specified in the key block header may be more specific than the Use values defined in this document. For consistency, the following table defines the corresponding Use value for each TR-31 key usage:

| TR-31 Value | TR-31 Mode(s) of use | Definition | use |
|-------------|----------------------|---|--|
| “B0” | “X” | BDK Base Derivation Key | keyDerKey |
| “B1” | “X” | DUKPT Initial Key (also known as IPEK) | keyDerKey** pinRemote function* crypt macing |
| “C0” | “C”, “G”, “V” | CVK Card Verification Key | NA |
| “D0” | “B”, “D”, “E” | Data Encryption using ecb, cbc, cfb, ofb, ccm or ctr | crypt |
| “E0” | “X” | EMV/chip Issuer Master Key: Client cryptograms | rsaPublicVerify |
| “E1” | “X” | EMV/chip Issuer Master Key: Secure Messaging for Confidentiality | rsaPublicVerify |
| “E2” | “X” | EMV/chip Issuer Master Key: Secure Messaging for Integrity | rsaPublicVerify |
| “E3” | “X” | EMV/chip Issuer Master Key: Data Authentication Code | rsaPublicVerify |
| “E4” | “X” | EMV/chip Issuer Master Key: Dynamic Numbers | rsaPublicVerify |
| “E5” | “X” | EMV/chip Issuer Master Key: Card Personalization | rsaPublicVerify |
| “E6” | “X” | EMV/chip Issuer Master Key: Other | rsaPublicVerify |
| “I0” | “N” | Initialization Vector (IV) | NA |
| “K0” | “B”, “D”, “E” | Key Encryption or wrapping | keyEncKey svEncKey |
| “K1” | “B”, “D”, “E” | TR-31 Key Block Protection Key | anstr31Master |
| “M0” | “C”, “G”, “V” | ISO 16609 MAC algorithm 1 (using TDEA) | macing |
| “M1” | “C”, “G”, “V” | ISO 9797-1 MAC Algorithm 1 | macing |

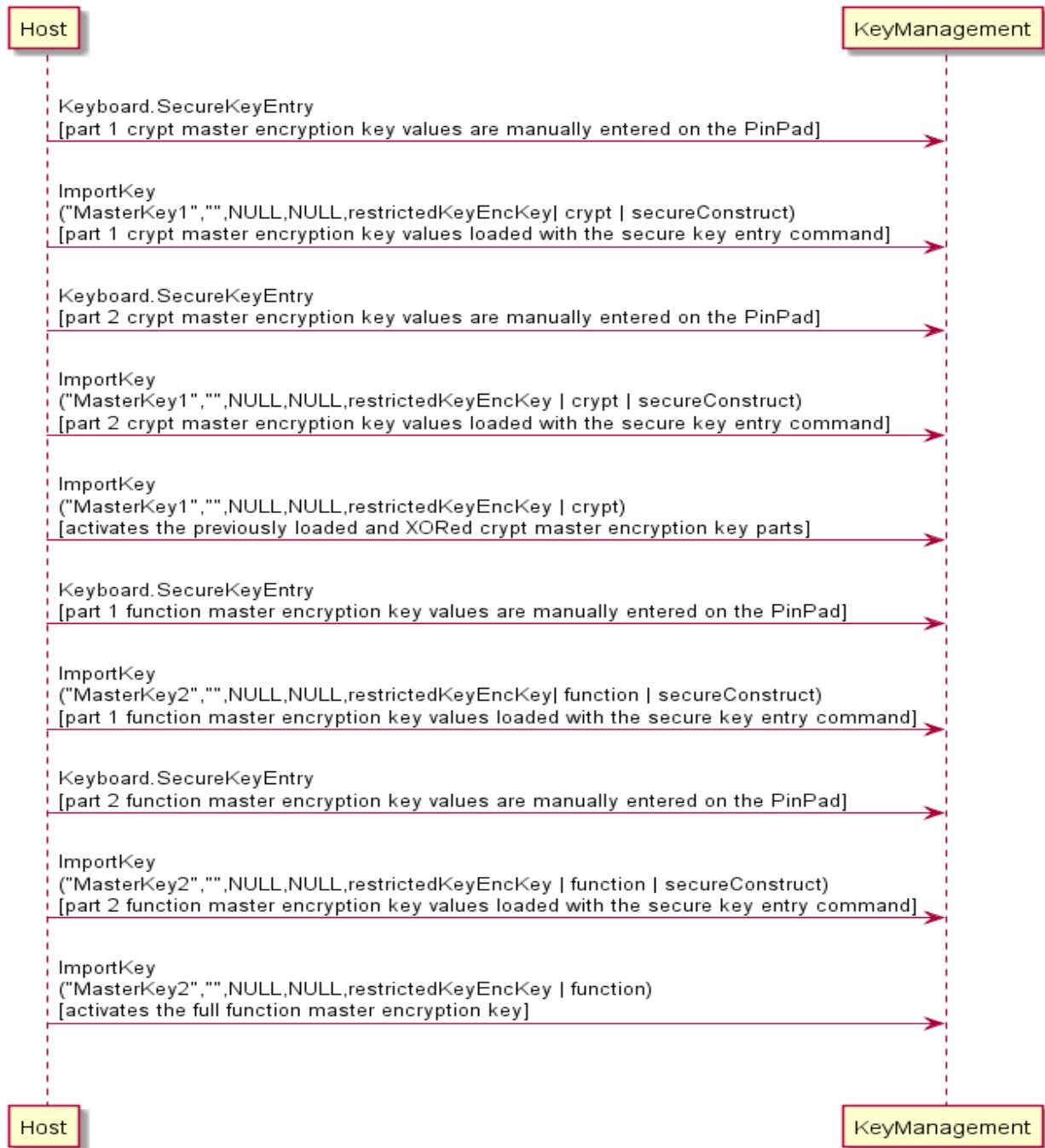
| | | | |
|------|---------------|--|---------------------|
| “M2” | “C”, “G”, “V” | ISO 9797-1 MAC Algorithm 2 | macing |
| “M3” | “C”, “G”, “V” | ISO 9797-1 MAC Algorithm 3 | macing |
| “M4” | “C”, “G”, “V” | ISO 9797-1 MAC Algorithm 4 | macing |
| “M5” | “C”, “G”, “V” | ISO 9797-1 MAC Algorithm 5 | macing |
| “P0” | “B”, “D”, “E” | PIN Encryption | pinRemote function* |
| “V0” | “C”, “G”, “V” | PIN verification, KPV, other algorithm | pinLocal function* |
| “V1” | “C”, “G”, “V” | PIN verification, IBM 3624 | pinLocal function* |
| “V2” | “C”, “G”, “V” | PIN Verification, VISA PVV | pinLocal function* |

*Note that function is listed here for backward compatibility, but pinLocal/pinRemote is the more accurate single-use value.

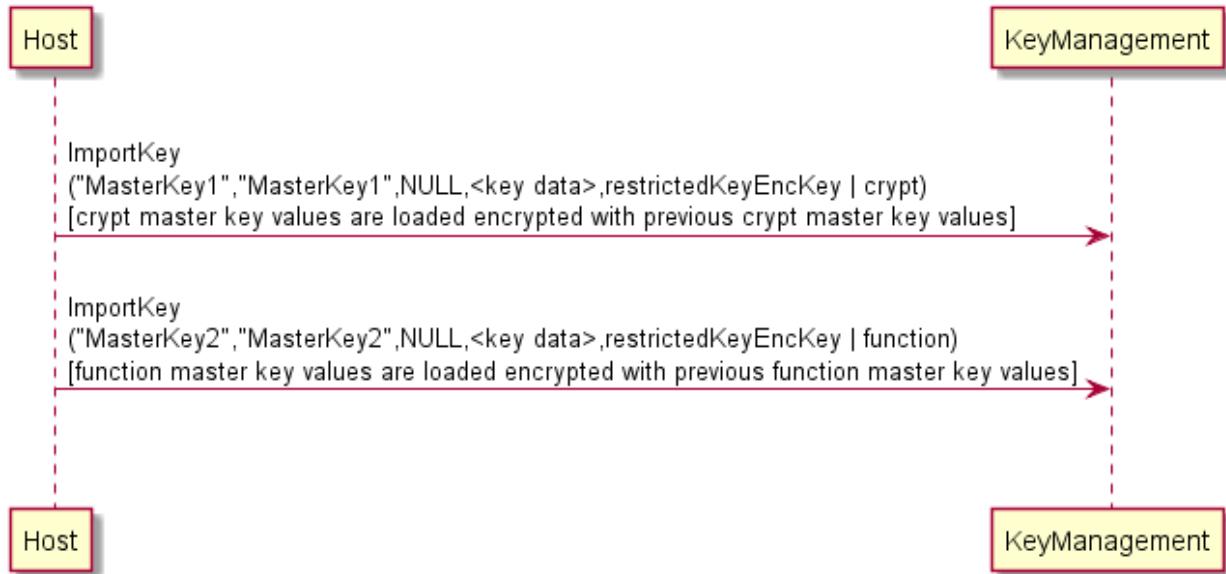
** The Base Derivation Key is used to derive the IPEK. When a dukpt IPEK is loaded, derived future keys are stored and the IPEK deleted. Therefore, while the IPEK is no longer loaded, future keys directly related to it are. pinRemote and optionally function are included as the primary use of an IPEK future key is to create a variant for PIN encryption. If the optional variant data encryption and MAC keys are supported, crypt and macing must be included. To use the optional data or MAC keys in a crypt command, key must be the name of the IPEK and Algorithm must be cryptTriDesCbc cryptTriDesMac. If the optional data encryption key is being used, Mode must be modeEncCrypt. The optional variant response data encryption and MAC keys are not supported.

7.2.22 - RestrictedKeyEndKey Command Usage

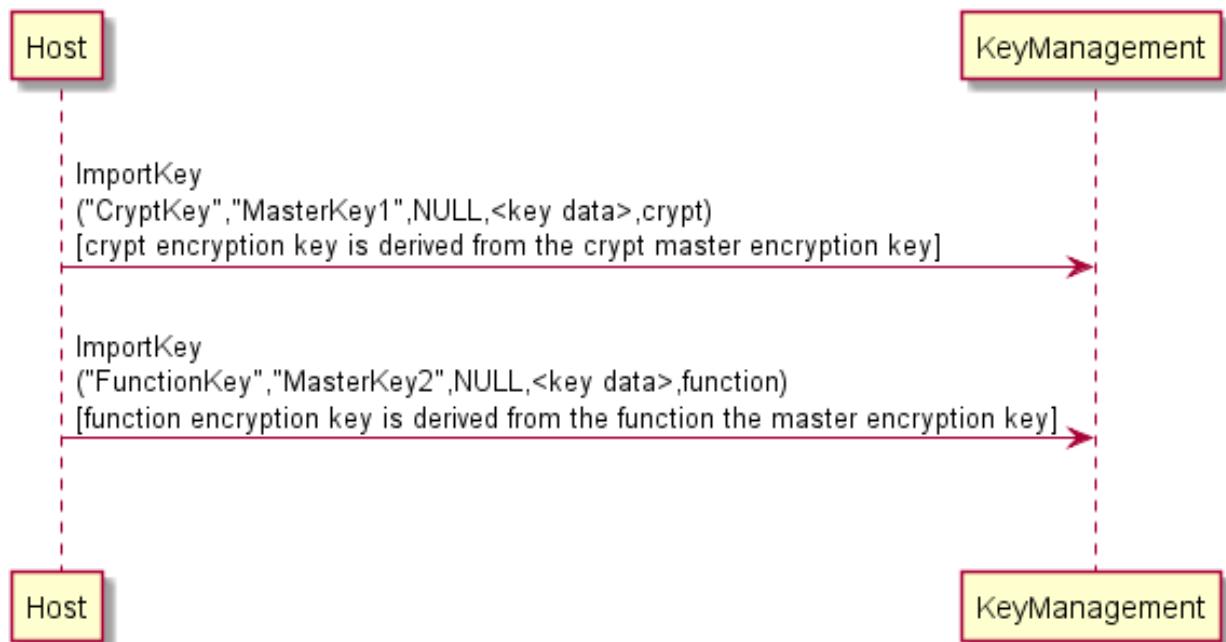
This sample command flow sequence shows how encryption keys can be derived/not derived if the master key has a restricted use. NOTE: In this example the master encryption key is loaded using the secure key entry command instead of using RKL commands. The loading with RKL works in the same way. Secure key entry based restricted master encryption key loading with RestrictedKeyEncKey flag:



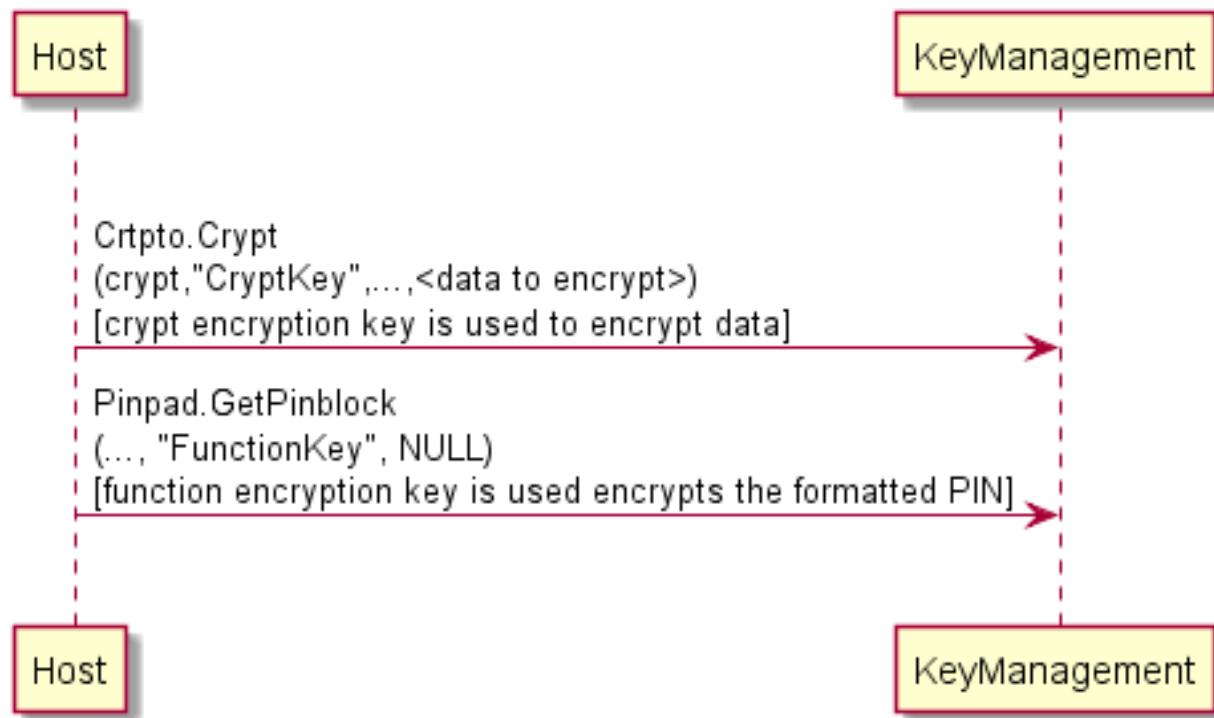
New master keys loaded with restrictedKeyEncKey flag, encrypted with themselves



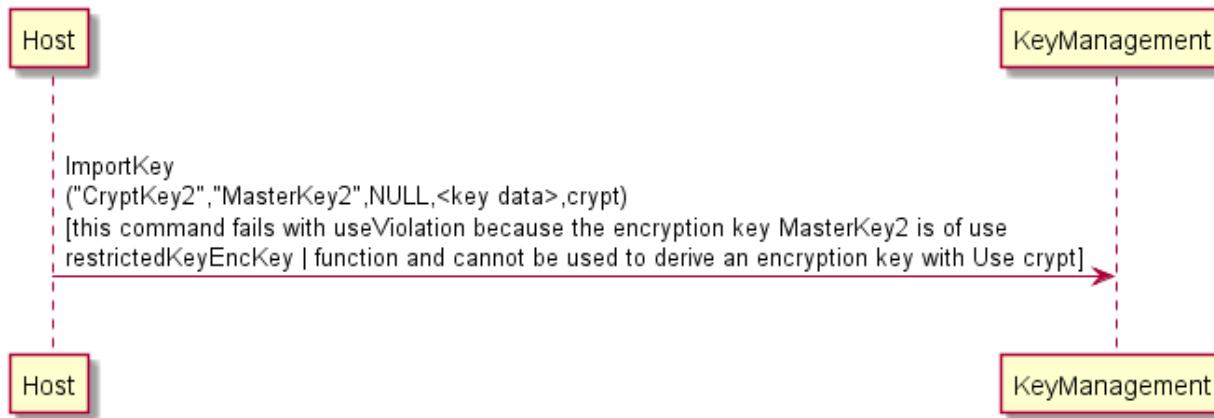
Loading derived keys:



Usage sample for derived keys



Master key restriction disallows loading of derived keys with different usage:



7.3 - Command Messages

7.3.1 - KeyManagement.GetKeyDetail

This command returns extended detailed information about the keys in the encryption module, including des, dukpt, aes, rsa private and public keys. This command will also return information on all keys loaded during manufacture that can be used by clients. Details

relating to the keys loaded using OPT (via the ZKA ProtIsoPs protocol) are retrieved using the ZKA hsmLdi protocol. These keys are not reported by this command.

Command Message

| Payload | Type | Required |
|--------------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "keyName": Add example to YAML | string | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

keyName

Name of the key for which detailed information is requested.

Completion Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "keyNotFound", | string | |
| "keyDetail": [{ | array (object) | |
| "keyName": Add example to YAML, | string | |
| "generation": 0, | integer | |

```

"version": 0,                                integer
"activatingDate": [0],                         array (integer)
"expiryDate": [0],                            array (integer)
"loaded": {                                    object
  "no": false,                                boolean
  "yes": false,                               boolean
  "unknown": false,                           boolean
  "construct": false,                         boolean
},
"keyBlockInfo": {                             object
  "keyUsage": Add example to YAML,           string
  "algorithm": Add example to YAML,          string
  "modeOfUse": Add example to YAML,          string
  "keyVersionNumber": Add example to YAML,   string
  "exportability": Add example to YAML,      string
  "optionalBlockHeader": Add example to YAML, string
  "keyLength": 0,                             integer
}
}
}

```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- keyNotFound - The specified key name is not found.

keyDetail/keyName

Specifies the name of the key.

keyDetail/generation

Specifies the generation of the key as bcd value. Different generations might correspond to different environments (e.g. test or production environment). The content is vendor specific. This value will be 0xFF if no such information is available for the key.

keyDetail/version

Specifies the version of the key (the year in which the key is valid, e.g. 01 for 2001) as bcd value. This value will be 0xFF if no such information is available for the key.

keyDetail/activatingDate

Specifies the date when the key is activated as bcd value in the format YYYYMMDD. This value will be 0xFFFFFFFF if no such information is available for the key.

keyDetail/expiryDate

Specifies the date when the key expires as bcd value in the format YYYYMMDD. This value will be 0xFFFFFFFF if no such information is available for the key.

keyDetail/loaded

Specifies whether the key has been loaded (imported from Client or locally from Operator).

keyDetail/loaded/no

The key is not loaded or not ready to be used in cryptographic operations.

keyDetail/loaded/yes

The key is loaded and ready to be used in cryptographic operations.

keyDetail/loaded/unknown

The State of the key is unknown.

keyDetail/loaded/construct

The key is under construction, meaning that at least one key part has been loaded but the key is not activated and ready to be used in other cryptographic operations. This flag can only be returned in combination with loaded_no.

keyDetail/keyBlockInfo

Contains the key block header of keys imported within an ANS TR-31 key block. This data is encoded in the same format that it was imported in, and contains all mandatory and optional header fields. keyBlockHeader is NULL if the key was not imported within a key block or has not been loaded yet. The use field provides an accurate summary of the key use, but the use defined within the key block header is more precise.

keyDetail/keyBlockInfo/keyUsage

Specifies the intended function of the key. See [Reference 35. ANS X9 TR-31 2018] for all possible values.

keyDetail/keyBlockInfo/algorithm

Specifies the algorithm for which the key may be used. See [Reference 35. ANS X9 TR-31 2018] for all possible values.

keyDetail/keyBlockInfo/modeOfUse

Specifies the operation that the key may perform. See [Reference 35. ANS X9 TR-31 2018] for all possible values.

keyDetail/keyBlockInfo/keyVersionNumber

Specifies a two-digit ASCII character version number, which is optionally used to indicate that contents of the key block are a component, or to prevent re-injection of old keys. See

[Reference 35. ANS X9 TR-31 2018] for all possible values.

keyDetail/keyBlockInfo/exportability

Specifies whether the key may be transferred outside of the cryptographic domain in which the key is found. See [Reference 35. ANS X9 TR-31 2018] for all possible values.

keyDetail/keyBlockInfo/optionalBlockHeader

Contains any optional header blocks, as defined in [Reference 35. ANS X9 TR-31 2018]. This value will be NULL if there are no optional block headers.

keyDetail/keyBlockInfo/keyLength

Specifies the length, in bits, of the key. 0 if the key length is unknown.

Event Messages

None

7.3.2 - KeyManagement.Initialization

The encryption module must be initialized before any encryption function can be used. Every call to [KeyManagement.Initialization](#) destroys all client keys that have been loaded or imported; it does not affect those keys loaded during manufacturing.

Usually this command is called by an operator task and not by the client program. Public keys imported under the rsa Signature based remote key loading scheme when public key deletion authentication is required will not be affected. However, if this command is requested in authenticated mode, public keys that require authentication for deletion will be deleted. This includes public keys imported under either the rsa Signature based remote key loading scheme or the TR34 RSA Certificate based remote key loading scheme.

Initialization also involves loading 'initial' client keys and local vendor dependent keys. These can be supplied, for example, by an operator through a keyboard, a local configuration file, remote RSA key management or possibly by means of some secure hardware that can be attached to the device. The client 'initial' keys would normally get updated by the client during a [KeyManagement.ImportKey](#) command as soon as possible. Local vendor dependent static keys (e.g. storage, firmware and offset keys) would normally be transparent to the client and by definition cannot be dynamically changed.

Where initial keys are not available immediately when this command is issued (i.e. when operator intervention is required), the Service Provider returns `accessDenied` and the client must await the [KeyManagement.InitializedEvent](#).

During initialization an optional encrypted ID key can be stored in the HW module. The ID key and the corresponding encryption key can be passed as parameters; if not, they are generated automatically by the encryption module. The encrypted ID is returned to the client and serves as authorization for the key import function. The [Capabilities](#) command indicates whether or not the device will support this feature.

This function also resets the hsm terminal data, except session key index and trace number.

This function resets all certificate data and authentication public/private keys back to their initial states at the time of production (except for those public keys imported under the rsa Signature based remote key loading scheme when public key deletion authentication is required). Key-pairs created with [KeyManagement.GenerateRSAKeyPair](#) are deleted. Any keys installed during production, which have been permanently replaced, will not be reset. Any Verification certificates that may have been loaded must be reloaded. The Certificate state will remain the same, but the [KeyManagement.LoadCertificate](#) or [KeyManagement.ReplaceCertificate](#) commands must be called again.

When multiple ZKA HSMs are present, this command deletes all keys loaded within all ZKA logical HSMs."

Command Message

| Payload | Type | Required |
|-------------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "ident": Add example to YAML, | string | |
| "key": Add example to YAML | string | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

ident

The value of the ID key. this field is not required if an indent is not required.

key

The value of the encryption key formatted in base64. this field is not required if no specific key name required.

Completion Message

| <u>Payload</u> | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "accessDenied", | string | |
| "identification": Add example to YAML | string | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- accessDenied - The encryption module is either not initialized or not ready for any vendor specific reason.

- invalidId - The ID passed was not valid.

identification

The value of the ID key encrypted by the encryption key formatted in base64. This value can be used as authorization for the [KeyManagement.ImportKey](#) command, this field is not set if no authorization required.

Event Messages

None

7.3.3 - KeyManagement.DeriveKey

The encryption key in the secure key buffer or passed by the client is loaded in the encryption module. The key can be passed in clear text mode or encrypted with an accompanying 'key encryption key'. A key can be loaded in multiple unencrypted parts by combining the construct or secureConstruct value with the final usage flags within the use field. If the construct flag is used then the client must provide the key data through the value parameter, If secureConstruct is used then the encryption key part in the secure key buffer previously populated with the [Keyboard.SecureKeyEntry](#) command is used and value is ignored. Key parts loaded with the secureConstruct flag can only be stored once as the encryption key in the secure key buffer is no longer available after this command has been executed. The construct and secureConstruct construction flags cannot be used in combination.

Command Message

| Payload | Type | Required |
|---------------------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "derivationAlgorithm": 0, | integer | |
| "key": Add example to YAML, | string | |
| "keyGenKey": Add example to YAML, | string | |
| "startValueKey": Add example to YAML, | string | |
| "startValue": Add example to YAML, | string | |

```
"padding": 0,           integer  
"inputData": Add example to YAML,    string  
"ident": Add example to YAML        string  
}
```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

derivationAlgorithm

Specifies the algorithm that is used for derivation.

key

Specifies the name where the derived key will be stored.

keyGenKey

Specifies the name of the key generating key that is used for the derivation.

startValueKey

Specifies the name of the stored key used to decrypt the startValue to obtain the Initialization Vector. If this field is not set, startValue is used as the Initialization Vector.

startValue

des initialization vector for the encryption step within the derivation. "

padding

Specifies the padding character for the encryption step within the derivation. The valid

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

range is 0x00 to 0xFF

inputData

Data to be used for key derivation.

ident

Specifies the key owner identification. It is a handle to the encryption module and is returned to the client in the [KeyManagement.Initialization](#) command. See [Capabilities.idkey](#) for whether this value is required. If not required, this field should not be set. The use of this parameter is vendor dependent.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "keyNotFound" | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- keyNotFound - The specified key was not found.
- accessDenied - The encryption module is either not initialized or not ready for any vendor specific reason.
- invalidId - The ID passed was not valid.
- duplicateKey - A key exists with that name and cannot be overwritten.
- keyNoValue - The specified key is not loaded.
- useViolation - The specified use is not supported by this key.
- invalidKeyLength - The length of startValue is not supported or the length of an encryption key is not compatible with the encryption operation required.
- algorithmNotSupported - The specified algorithm is not supported.

Event Messages

None

[7.3.4 - KeyManagement.Reset](#)

Sends a service reset to the Service Provider.

Command Message

| Payload | Type | Required |
|---------------------------|---------|----------|
| { "timeout": 5000 } | integer | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|---------|------|----------|
|---------|------|----------|

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

```
{
  "completionCode": "success",           string
  "errorDescription": Add example to YAML  string
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

Event Messages

None

[7.3.5 - KeyManagement.ImportKey](#)

The encryption key passed by the client is loaded in the encryption module. For secret keys, the key must be passed encrypted with an accompanying "key encrypting key" or "key block protection key". For public keys, they key is not required to be encrypted but is required to have verification data in order to be loaded.

Command Message

| Payload | Type | Required |
|-----------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "key": Add example to YAML, | string | |
| "keyAttributes": { | object | |

```

"keyUsage": Add example to YAML,           string
"algorithm": Add example to YAML,          string
"modeOfUse": Add example to YAML,          string
"cryptoMethod": Add example to YAML       string
},
"value": Add example to YAML,             string
"decryptKey": Add example to YAML,        string
"decryptMethod": "ecb",                  string
"verificationData": Add example to YAML, string
"verifyKey": Add example to YAML,         string
"verifyAttributes": {
"algorithm": Add example to YAML,        string
"cryptoMethod": "kcvNone",               string
"hashAlgorithm": "sha1"                 string
},
"vendorAttributes": Add example to YAML  string
}

```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

key

Specifies the name of key being loaded.

keyAttributes

This parameter specifies the encryption algorithm, cryptographic method, and mode to be used for the key imported by this command. For a list of valid values see the [KeyManagement.keyAttribute](#) capability. The values specified must be compatible with the key identified by key.

keyAttributes/keyUsage

Specifies the Key usages supported by [KeyManagement.ImportKey](#) command and key usage string length must be two. The following values are possible:

- B0 - BDK Base Derivation Key.
- B1 - Initial DUKPT key.
- B2 - Base Key Variant Key.
- C0 - CVK Card Verification Key.
- D0 - Symmetric Key for Data Encryption.
- D1 - Asymmetric Key for Data Encryption.
- D2 - Data Encryption Key for Decimalization Table.
- E0 - SMV/Chip Issuer MAster Key: Client Cryptogram.
- E1 - EMV/Chip Issuer Master Key: Secure Messaging for Confidentiality.
- E2 - EMV/chip Issuer Master Key: Secure Messaging for Integrity.
- E3 - EMV/chip Issuer Master Key: Data Authentication Code.
- E4 - EMV/chip Issuer MAster Key: Dynamic.
- E5 - EMV/Chip Issuer MAster Key: Card Personalization.
- E6 - EMV/chip Issuer MAster Key: Other Initialization Vector(IV).
- I0 - Initialization Vector(IV).
- K0 - Key Encryption or wrapping.
- K1 - TR-31 Key Block Protection Key.
- K2 - TR-34 Asymmetric Key.
- K3 - Asymmetric Key for key agreement/key wrapping.
- M0 - ISO 16609 MAC algorithm 1(using TDEA).
- M1 - ISO 9797-1 MAC Algorithm 1.
- M2 - ISO 9797-1 MAC Algorithm 2.
- M3 - ISO 9797-1 MAC Algorithm 3.
- M4 - ISO 9797-1 MAC Algorithm 4.
- M5 - ISO 9797-1:2011 MAC Algorithm.
- M6 - ISO 9797-1:2011 MAC Algorithm 5/CMAC.
- M7 - HMAC.
- M8 - ISO9797-1:2011 MAC Algorithm 6.
- P0 - PIN Encryption.
- S0 - Asymmetric key pair for digital signature.
- S1 - Asymmetric key pair , CA key.

- S2 - Asymmetric key pair, nonX9.24 key.
- V0 - PIN verification, KPV, other algorithm.
- V1 - PIN verification, IBM 3624.
- V2 - PIN verification,VISA PVV.
- V3 - PIN verification,X9-132 algorithm 1.
- V4 - PIN verification, X9-132 algorithm 2.
- 0B - Restricted Key Encryption Key that can only be used to load DUKPT keys.
- 0C - Restricted Key Encryption Key that can only be used to load CVKs.
- 0E - Restricted Key Encryption Key that can only be used to load EMV keys.
- 0I - Restricted Key Encryption Key that can only be used to load Initialization Vectors.
- 0K - Restricted Key Encryption Key that can be only used to load keys that can load other keys.
- 0M - Restricted Key Encryption Key that can only be used to load Initialization Vectors.
- 0P - Restricted Key Encryption Key that can only be used to load PIN encryption keys.
- 0S - Restricted Key Encryption Key that can only be used to load asymmetric key pairs.
- 0V - Restricted Key Encryption Key that can only be used to load PIN verification keys.
- 1B - Restricted Keyblock Protection Key that can only be used to load DUKPT keys.
- 1C - Restricted Key Keyblock Protection Key that can only be used to load CVKs.
- 1D - Restricted Keyblock Protection Key that can only be used to load data encryption keys.
- 1E - Restricted Keyblock Protection Key that can only be used to load EMV keys.
- 1I - Restricted Keyblock Protection Key that can only be used to load Initialization Vectors.
- 1K - Restricted Keyblock Protection Key that can only be used to load keys that can load other keys.
- 1M - Restricted Keyblock Protection Key that can only be used to load MAC keys.
- 1P - Restricted Keyblock Protection Key that can only be used to load PIN encryption keys.
- 1S - Restricted Keyblock Protection Key that can only be used to load asymmetric key pairs.
- 1V - Restricted Keyblock Protection Key that can only be used to load PIN verification keys.
- "00" - "99" - These numeric values are reserved for proprietary use.

keyAttributes/algorithm

Specifies the encryption algorithms supported by the import command. The following

values are possible:

- A - AES. * D - DEA. * R - RSA. * T - Triple DEA (also referred to as TDEA). * "0" - "9" - These numeric values are reserved for proprietary use.

keyAttributes/modeOfUse

Specifies the encryption modes supported by import key. The following values are possible:

- B - Both Encrypt and Decrypt/ Wrap and unwrap.
- C - Both Generate and Verify.
- D - Decrypt/ Unwrap Only.
- E - Encrypt/ Wrap Only.
- G - Generate Only.
- S - Signature Only.
- T - Both Sign and Decrypt.
- V - Verify Only.
- X - Key used to derive other keys(s).
- Y - Key used to create key variants.
- "0" - "9" - These numeric values are reserved for proprietary use.

keyAttributes/cryptoMethod

Specifies the cryptographic methods supported by the import command. For attributes, this parameter is 0, because the key being imported is not being used yet to perform a cryptographic method.

value

Specifies the value of key to be loaded formatted in base64. If it is an rsa key the first 4 bytes contain the exponent and the following 128 the modulus.

decryptKey

Specifies the name of the key used to decrypt the key being loaded. If value contains a TR-31 key block, then decryptKey is the name of the key block protection key that is used to verify and decrypt the key block. This property is not required if the data in Value is not encrypted.

decryptMethod

Specifies the cryptographic method that shall be used with the key specified by decryptKey. The device shall use this method to decrypt the encrypted value in the value parameter.

This property is not required if a keyblock is being imported, as the decrypt method is contained within the keyblock. This parameter specifies the cryptographic method that will be used with the encryption algorithm specified by algorithm.

For a list of valid values see the cryptoMethod property in the [KeyManagement.decryptAttribute](#) capability. If algorithm is 'A', 'D', or 'T', then this property decryptMethod can be one of the following values:

- ecb - The ECB encryption method.
- cbc - The CBC encryption method.
- cfb - The CFB encryption method.
- ofb - The OFB encryption method.
- ctr - The CTR method defined in NIST SP800-38A.
- xts - The XTS method defined in NIST SP800-38E.

If algorithm is 'R', then this property decryptMethod can be one of the following values:

- rsaesPkcs1V15 - Use the RSAES_PKCS1-v1.5 algorithm.
- rsaesOaep - Use the RSAES OAEP algorithm.

If keyUsage is 'K1', then this property decryptMethod is not set. TR-31 defines the cryptographic methods used for each key block version.

verificationData

Contains the data to be verified before importing. This property is not required if no verification is needed before importing the key.

verifyKey

Specifies the name of the previously loaded key which will be used to verify the verificationData. This property is not required when no verification is needed before importing the key.

verifyAttributes

This parameter specifies the encryption algorithm, cryptographic method, and mode to be used to verify this command or to generate verification output data. Verifying input data will result in no verification output data. For a list of valid values see the [Capabilities.verifyAttributes](#) capability. This property must not be set if verificationData is not required.

verifyAttributes/algorithm

Specifies the encryption algorithms supported by the import command. The following

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

values are possible:

- A - AES.
- D - DEA.
- R - RSA.
- T - Triple DEA (also referred to as TDEA).
- "0" - "9" - These numeric values are reserved for proprietary use.

verifyAttributes/cryptoMethod

This parameter specifies the cryptographic method that will be used with encryption algorithm.

If algorithm is 'A', 'D', or 'T' and keyUsage is a MAC usage (i.e. 'M1'), then this property cryptoMethod not be set.

If algorithm is 'A', 'D', or 'T' and keyUsage is '00', then this property cryptoMethod can be one of the following values:

- kcvNone - There is no key check value verification required.
- kcvSelf - The key check value (KCV) is created by an encryption of the key with itself.
- kcvZero - The key check value (KCV) is created by encrypting a zero value with the key.

If algorithm is 'R' and keyUsage is not '00', then this property cryptoMethod can be one of the following values:

- sigNone - No signature algorithm specified. No signature verification will take place and the content of verificationData must not be set.
- rsassaPkcs1V15 - Use the RSASSA-PKCS1-v1.5 algorithm.
- rsassaPss - Use the RSASSA-PSS algorithm.

verifyAttributes/hashAlgorithm

For asymmetric signature verification methods (keyUsage is 'S0', 'S1', or 'S2'), this can be one of the following values to be used. If keyUsage is specified as any of the MAC usages (i.e. 'M1'), then properties should not be not set or both 'sha1' and 'sha256' are false.

- sha1 - The SHA 1 digest algorithm.
- sha256 - The SHA 256 digest algorithm, as defined in ISO/IEC 10118-3:2004 and FIPS 180-2.

vendorAttributes

Specifies the vendor attributes of the key to be imported. Refer to vendor documentation

for details. If no vendor attributes are used, then this property must not be set.

Completion Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "keyNotFound", | string | |
| "verificationData": Add example to YAML, | string | |
| "verifyAttributes": { | object | |
| "keyUsage": Add example to YAML, | string | |
| "algorithm": Add example to YAML, | string | |
| "modeOfUse": Add example to YAML, | string | |
| "cryptoMethod": "kcvNone", | string | |
| "hashAlgorithm": { | object | |
| "sha1": false, | boolean | |
| "sha256": false | boolean | |
| } | | |
| }, | | |
| "keyLength": 0 | integer | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

information

errorCode

Specifies the error code if applicable. The following values are possible:

- keyNotFound - One of the keys specified was not found.
- accessDenied - The encryption module is either not initialized or not ready for any vendor specific reason.
- duplicateKey - A key exists with that name and cannot be overwritten.
- keyNoValue - One of the specified keys is not loaded.
- useViolation - The use specified by keyUsage is not supported or conflicts with a previously loaded key with the same name as key.
- formatNotSupported - The specified format is not supported.
- invalidKeyLength - The length of value is not supported.
- noKeyRam - There is no space left in the key RAM for a key of the specified type.
- signatureNotSupported - The cryptoMethod of the verifyAttributes is not supported. The key is not stored in the PIN.
- signatureInvalid - The verification data in the input data is invalid. The key is not stored in the PIN.
- randomInvalid - The encrypted random number in the input data does not match the one previously provided by the PIN device. The key is not stored in the PIN.
- algorithmNotSupported - The algorithm specified by bAlgorithm is not supported by this command.
- modeNotSupported - The mode specified by modeOfUse is not supported.
- cryptoMethodNotSupported - The cryptographic method specified by cryptoMethod for keyAttributes or verifyAttributes is not supported.

verificationData

The verification data. This field is not set if there is no verification data.

verifyAttributes

This parameter specifies the encryption algorithm, cryptographic method, and mode used to verify this command. For a list of valid values see the [Capabilities.verifyAttributes](#) capability fields. This field is not set if there is no verification data.

verifyAttributes/keyUsage

Specifies the key usages supported by the import command. The following values are

possible:

- M0 - ISO 16609 MAC Algorithm 1 (using TDEA).
- M1 - ISO 9797-1 MAC Algorithm 1.
- M2 - ISO 9797-1 MAC Algorithm 2.
- M3 - ISO 9797-1 MAC Algorithm 3.
- M4 - ISO 9797-1 MAC Algorithm 4.
- M5 - ISO 9797-1:1999 MAC Algorithm 5.
- M6 - ISO 9797-1:2011 MAC Algorithm 5/CMAC.
- M7 - HMAC.
- M8 - ISO9797-1:2011 MAC Algorithm 6.
- S0 - Asymmetric key pair or digital signature.
- S1 - Asymmetric key pair, CA key.
- S2 - Asymmetric key pair, nonX9.24 key.
- "00" - "99" - These numeric values are reserved for proprietary use.

verifyAttributes/algorithm

Specifies the encryption algorithms supported by the import command. The following values are possible:

- A - AES.
- D - DEA.
- R - RSA.
- T - Triple DEA (also referred to as TDEA).
- "0" - "9" - These numeric values are reserved for proprietary use.

verifyAttributes/modeOfUse

Specifies the encryption modes supported by the import command. The following values are possible:

- V - Verify Only.
- "0" - "9" - These numeric values are reserved for proprietary use.

verifyAttributes/cryptoMethod

This parameter specifies the cryptographic method that will be used with encryption algorithm.

If algorithm is 'A', 'D', or 'T' and keyUsage is a MAC usage (i.e. 'M1'), then this property cryptoMethod not be set.

If algorithm is 'A', 'D', or 'T' and keyUsage is '00', then this property cryptoMethod can be

one of the following values:

- kcvNone - There is no key check value verification required.
- kcvSelf - The key check value (KCV) is created by an encryption of the key with itself.
- kcvZero - The key check value (KCV) is created by encrypting a zero value with the key.

If algorithm is 'R' and keyUsage is not '00', then this property cryptoMethod can be one of the following values:

- sigNone - No signature algorithm specified. No signature verification will take place and the content of verificationData must not be set.
- rsassaPkcs1V15 - Use the RSASSA-PKCS1-v1.5 algorithm.
- rsassaPss - Use the RSASSA-PSS algorithm.

verifyAttributes/hashAlgorithm

For asymmetric signature verification methods (keyUsage is 'S0', 'S1', or 'S2'), this can be one of the following values to be used. If keyUsage is specified as any of the MAC usages (i.e. 'M1'), then properties should not be not set or both 'sha1' and 'sha256' are false.

verifyAttributes/hashAlgorithm/sha1

The SHA 1 digest algorithm.

verifyAttributes/hashAlgorithm/sha256

The SHA 256 digest algorithm, as defined in ISO/IEC 10118-3:2004 and FIPS 180-2.

keyLength

Specifies the length, in bits, of the key. 0 is the key length is unknown.

Event Messages

None

[7.3.6 - KeyManagement.DeleteKey](#)

This command can be used to delete a key without authentication. Where an authenticated delete is required, the [KeyManagement.StartAuthenticate](#) command should be used.

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Command Message

| Payload | Type | Required |
|----------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "key": Add example to YAML | string | |
| } | | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

key

Specifies the name of key being deleted. if this field is not specified or an empty string, all keys are deleted.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "accessDenied" | string | |
| } | | |

Properties**completionCode**

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- accessDenied - The encryption module is either not initialized or not ready for any vendor specific reason.

Event Messages

None

7.3.7 - KeyManagement.ExportRSAIssuerSignedItem

This command is used to export data elements from the device, which have been signed by an offline Signature Issuer. This command is used when the default keys and Signature Issuer signatures, installed during manufacture, are to be used for remote key loading.

This command allows the following data items are to be exported:

- The Security Item which uniquely identifies the device. This value may be used to uniquely identify a device and therefore confer trust upon any key or data obtained from this device.
- The rsa public key component of a public/private key pair that exists within the device. These public/private key pairs are installed during manufacture. Typically, an exported public key is used by the host to encipher the symmetric key.

Command Message

| Payload | Type | Required |
|----------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "exportItemType": "eppId", | string | |

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

"**name**": Add example to YAML string

}

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

exportItemType

Defines the type of data item to be exported from the device.

name

Specifies the name of the public key to be exported. The private/public key pair was installed during manufacture; see section 8.1.8 (Default Keys and Security Item loaded during manufacture) for a definition of these default keys. If name is an empty string, then the default EPP public key that is used for symmetric key encryption is exported.

Completion Message

| Payload | Type | Required |
|---|--------|----------|
| { | | |
| " completionCode ": "success", | string | |
| " errorDescription ": Add example to YAML, | string | |
| " errorCode ": "noRSAKeyPair", | string | |
| " value ": Add example to YAML, | string | |
| " rsaSignatureAlgorithm ": "na", | string | |
| " signature ": Add example to YAML | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- noRSAKeyPair - The PIN device does not have a private key.
- accessDenied - The encryption module is either not initialized or not ready for any vendor specific reason.
- keyNotFound - The data item identified by name was not found.

value

If a public key was requested then value contains the PKCS #1 formatted rsa public key represented in DER encoded ASN.1 format. If the security item was requested then value contains the PIN's Security Item, which may be vendor specific.

rsaSignatureAlgorithm

Specifies the algorithm used to generate the Signature returned in signature

signature

Specifies the RSA signature of the data item exported formatted in base64. An empty sting can be returned when key signature are not supported.

Event Messages

None

7.3.8 - KeyManagement.GenerateRSAKeyPair

This command will generate a new rsa key pair. The public key generated as a result of this command can subsequently be obtained by calling

[KeyManagement.ExportRSAEPPSignedItem](#). The newly generated key pair can only be used for the use defined in the use flag. This flag defines the use of the private key; its public key can only be used for the inverse function.

Command Message

| Payload | Type | Required |
|-----------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "key": Add example to YAML, | string | |
| "use": "rsaPrivate", | string | |
| "modulusLength": 0, | integer | |
| "exponentValue": "default" | string | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

key

Specifies the name of the new key-pair to be generated. Details of the generated key-pair can be obtained through the [KeyManagement.KeyDetail](#) command.

use

Specifies what the private key component of the key pair can be used for. The public key part can only be used for the inverse function. For example, if the rsaPrivateSign use is specified, then the private key can only be used for signature generation and the partner

public key can only be used for verification.

modulusLength

Specifies the number of bits for the modulus of the rsa key pair to be generated. When zero is specified then the device will be responsible for defining the length.

exponentValue

Specifies the value of the exponent of the rsa key pair to be generated

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "accessDenied" | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- accessDenied - The encryption module is either not initialized or not ready for any vendor specific reason.

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

- invalidModulusLength - The modulus length specified is invalid.
- useViolation - The specified use is not supported by this key.
- duplicateKey - A key exists with that name and cannot be overwritten.
- keyGenerationError - The EPP is unable to generate a key pair.

Event Messages

None

7.3.9 - KeyManagement.ExportRSAEPPSignedItem

This command is used to export data elements from the device that have been signed by a private key within the epp. This command is used in place of the [KeyManagement.ExportRSAIssuerSignedItem](#) command, when a private key generated within the device is to be used to generate the signature for the data item. This command allows an client to define which of the following data items are to be exported.

- The Security Item which uniquely identifies the device. This value may be used to uniquely identify a device and therefore confer trust upon any key or data obtained from this device.
- The rsa public key component of a public/private key pair that exists within the device.

Command Message

| Payload | Type | Required |
|--------------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "exportItemType": "eppId", | string | |
| "name": Add example to YAML, | string | |
| "sigKey": Add example to YAML, | string | |
| "signatureAlgorithm": "na" | string | |
| } | | |

Properties

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

exportItemType

Defines the type of data item to be exported from the device

name

Specifies the name of the public key to be exported. This can either be the name of a key-pair generated through [KeyManagement.GenerateRsaKeyPair](#) or the name of one of the default key-pairs installed during manufacture.

sigKey

Specifies the name of the private key to use to sign the exported item.

signatureAlgorithm

Specifies the algorithm to use to generate the Signature returned in both the selfSignature and signature fields.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "noRSAKeyPair", | string | |
| "value": Add example to YAML, | string | |
| "selfSignature": Add example to YAML, | string | |
| "signature": Add example to YAML | string | |

}

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- noRSAKeyPair - The PIN device does not have a private key.
- accessDenied - The encryption module is either not initialized or not ready for any vendor specific reason.
- keyNotFound - The data item identified by name was not found.

value

If a public key was requested then value contains the pkcs #1 formatted rsa Public Key represented in DER encoded ASN.1 format. If the security item was requested then value contains the PIN's Security Item, which may be vendor specific.

selfSignature

If a public key was requested then selfSignature contains the rsa signature of the public key exported, generated with the key-pair's private component. An empty string can be returned when key selfSignatures are not supported/required.

signature

Specifies the rsa signature of the data item exported. An empty string can be returned when signatures are not supported/required.

Event Messages

None

7.3.10 - KeyManagement.GetCertificate

This command is used to read out the encryptor's certificate, which has been signed by the trusted Certificate Authority and is sent to the host. This command only needs to be called once if no new Certificate Authority has taken over. The output of this command will specify in the pkcs #7 message the resulting Primary or Secondary certificate.

Command Message

| Payload | Type | Required |
|----------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "getCertificate": "enckey" | string | |
| } | | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0**getCertificate**

Specifies which public key certificate is requested. If the [KeyManagement.Status](#) command indicates Primary Certificates are accepted, then the Primary Public Encryption Key or the Primary Public Verification Key will be read out. If the [KeyManagement.Status](#) command indicates Secondary Certificates are accepted, then the Secondary Public Encryption Key or the Secondary Public Verification Key will be read out.

Completion Message

| Payload | Type | Required |
|---------|------|----------|
|---------|------|----------|

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

```
{  
  "completionCode": "success",          string  
  "errorDescription": Add example to YAML, string  
  "errorCode": "accessDenied",          string  
  "certificate": Add example to YAML     string  
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- accessDenied - The encryption module is either not initialized or not ready for any vendor specific reason.
- invalidCertificateState - The certificate module is in a state in which the request is invalid.
- keyNotFound - The specified public key was not found.

certificate

Contains the certificate that is to be loaded represented in DER encoded ASN.1 notation. This data should be in a binary encoded pkcs #7 using the degenerate certificate only case of the signed-data content type in which the inner content's data file is omitted and there are no signers.

Event Messages

None

7.3.11 - KeyManagement.ReplaceCertificate

This command is used to replace the existing primary or secondary Certificate Authority certificate already loaded into the KeyManagement. This operation must be done by an Initial Certificate Authority or by a Sub-Certificate Authority. These operations will replace either the primary or secondary Certificate Authority public verification key inside of the KeyManagement. After this command is complete, the client should send the [KeyManagement.LoadCertificate](#) and [KeyManagement.GetCertificate](#) commands to ensure that the new HOST and the encryptor have all the information required to perform the remote key loading process.

Command Message

| Payload | Type | Required |
|---|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "replaceCertificate": Add example to YAML | string | |
| } | | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0**replaceCertificate**

The pkcs # 7 message that will replace the current Certificate Authority formatted in base64. The outer content uses the signedData content type, the inner content is a degenerate certificate only content containing the new ca certificate and Inner Signed Data type The certificate should be in a format represented in DER encoded ASN.1 notation.

Completion Message

| Payload | Type | Required |
|---|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "accessDenied", | string | |
| "newCertificateData": Add example to YAML | string | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- accessDenied - The encryption module is either not initialized or not ready for any vendor specific reason.
- formatInvalid - The format of the message is invalid.
- invalidCertificateState - The certificate module is in a state in which the request is invalid.

newCertificateData

A pkcs #7 using a Digest-data content type formatted in base64. The digest parameter should contain the thumb print value.

Event Messages

None

7.3.12 - KeyManagement.StartKeyExchange

This command is used to start communication with the host, including transferring the host's Key Transport Key, replacing the Host certificate, and requesting initialization remotely. This output value is returned to the host and is used in the

[KeyManagement.ImportKey](#) and

[KeyManagement.LoadCertificate](#)

to verify that the encryptor is talking to the proper host.

The [KeyManagement.ImportKey](#) command ends the key exchange process.

Command Message

| Payload | Type | Required |
|--------------------------------------|------|----------|
| { "timeout": 5000 integer } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|---|------|----------|
| { "completionCode": "success", string | | |

```
"errorDescription": Add example to YAML,  string  
"errorCode": "accessDenied",          string  
"randomItem": Add example to YAML      string  
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- accessDenied - The encryption module is either not initialized or not ready for any vendor specific reason.

randomItem

A randomly generated number created by the encryptor formatted in base 64. If the device does not support random number generation and verification, a zero length random number is returned and an empty string is returned.

Event Messages

None

[7.3.13 - KeyManagement.GenerateKCV](#)

This command returns the Key Check Value (kcv) for the specified key.

Command Message

| Payload | Type | Required |
|-----------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "key": Add example to YAML, | string | |
| "keyCheckMode": "self" | string | |
| } | | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

key

Specifies the name of key that should be used to generate the kcv.

keyCheckMode

Specifies the mode that is used to create the key check value.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "keyNotFound", | string | |
| "kcv": Add example to YAML | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- keyNotFound - The specified key encryption key was not found.
- keyNoValue - The specified key exists but has no value loaded.
- accessDenied - The encryption module is either not initialized or not ready for any vendor specific reason.
- modeNotSupported - The KCV mode is not supported.

kcv

Contains the key check value data that can be used for verification of the key formatted in base64.

Event Messages

None

[7.3.14 - KeyManagement.LoadCertificate](#)

This command is used to load a host certificate to make remote key loading possible. This command can be used to load a host certificate when there is not already one present in the encryptor as well as replace the existing host certificate with a new host certificate. The type of certificate (Primary or Secondary) to be loaded will be embedded within the actual certificate structure.

Command Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "loadOption": "newHost", | string | |
| "signer": "certHost", | string | |
| "certificateData": Add example to YAML | string | |
| } | | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

loadOption

Specifies the method to use to load the certificate

signer

Specifies the signer of the certificate to be loaded

certificateData

The structure that contains the certificate that is to be loaded represented in DER encoded ASN.1 notation. For loadNewHost, this data should be in a binary encoded PKCS #7 using the 'degenerate certificate only' case of the signed-data content type in which the inner content's data file is omitted and there are no signers. For replaceHost, the message has an outer signedData content type with the signerInfo encryptedDigest field containing the signature of signer. The inner content is binary encoded pkcs#7 using the degenerate certificate. The optional crl field may or may not be included in the pkcs#7 signedData structure.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "accessDenied", | string | |
| "rsaKeyCheckMode": "none", | string | |
| "rsaData": Add example to YAML | string | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- accessDenied - The encryption module is either not initialized or not ready for any vendor specific reason.
- formatInvalid - The format of the message is invalid.
- invalidCertificateState - The certificate module is in a state in which the request is invalid.

rsaKeyCheckMode

Defines algorithm/method used to generate the public key check value/thumb print. The check value can be used to verify that the public key has been imported correctly.

rsaData

A pkcs #7 structure using a Digest-data content type formatted in base64. The digest parameter should contain the thumb print value calculated by the algorithm specified by rsaKeyCheckMode. If rsaKeyCheckMode is none, then this field is not be set or an empty string.

Event Messages

None

7.3.15 - KeyManagement.StartAuthenticate

This command is used to retrieve the data that needs to be signed and hence provided to the Authenticate command in order to perform an authenticated action on the device. If this command returns data to be signed then the Authenticate command must be used to call the command referenced by startAuthenticate. Any attempt to call the referenced command without using the Authenticate command, if authentication is required, shall result in AuthRequired.

Command Message

| Payload | Type | Required |
|-----------------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "commandId": Add example to YAML, | string | |
| "inpData": { | object | |
| } | | |
| } | | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

commandId

The command name to which authentication is being applied

inputData

A payload to the input data of the command referred to by the execution command.

Completion Message

| Payload | Type | Required |
|---|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "internalCmdResult": Add example to YAML, | string | |
| "dataToSign": Add example to YAML, | string | |
| "signers": { | object | |
| "none": false, | boolean | |
| "certhost": false, | boolean | |
| "sighost": false, | boolean | |
| "ca": false, | boolean | |
| "hl": false, | boolean | |
| "tr34": false, | boolean | |
| "cbcmac": false, | boolean | |
| "cmac": false, | boolean | |
| "reserved_1": false, | boolean | |
| "reserved_2": false, | boolean | |
| "reserved_3": false | boolean | |

}

}

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

internalCmdResult

Result from the command referenced by execution command. If the data within payload is invalid or cannot be used for some reason, then hInternalCmdResult will return an error but the result of this command will be ok.

dataToSign

The data that must be signed by one of the authorities indicated by signers before the command referenced by execution command can be executed. If the command specified by execution command does not require authentication, then this field is not set string and the command result is success.

signers

Specifies the allowed signers of the data as a combination.

signers/none

Authentication is not required.

signers/certhost

The data is signed by the current Host, using the RSA certificate-based scheme.

signers/sighost

The data is signed by the current Host, using the RSA signature-based scheme.

signers/ca

The data is signed by the Certificate Authority (CA).

signers/hl

The data is signed by the Higher Level (HL) Authority.

signers/tr34

The format of the data that was signed complies with the data defined in X9 TR342012 [Ref. 42]. This value can only be used in combination with the CERTHOST, CA or HL flags.

signers/cbcmac

A MAC is calculated over the data using lpsKey and the CBC MAC algorithm.

signers/cmac

A MAC is calculated over the data using lpsKey and the CMAC algorithm.

signers/reserved_1

Reserved for a vendor-defined signing method.

signers/reserved_2

Reserved for a vendor-defined signing method.

signers/reserved_3

Reserved for a vendor-defined signing method.

Event Messages

None

7.4 - Unsolicited Messages

7.4.1 - KeyManagement.InitializedEvent

This event specifies that, as a result of a Initialization command, the encryption module is now initialized and the master key (where required) and any other initial keys are loaded; ready to import other keys

| Payload | Type | Required |
|-------------------------------|--------|----------|
| { | | |
| "ident": Add example to YAML, | string | |
| "key": Add example to YAML | string | |
| } | | |

Properties

ident

The value of the ID key formatted in base 64. if not required, this field can not be set or an empty string

key

The value of the encryption key formatted in base 64. if not required, this field can not be set or an empty string

7.4.2 - KeyManagement.IllegalKeyAccessEvent

This event specifies that an error occurred accessing an encryption key. Possible situations for generating this event are listed in the description of lErrorCode.

| Payload | Type | Required |
|---------|------|----------|
| { | | |

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

```
"keyName": Add example to YAML,  string  
"errorCode": "keynotfound"      string  
}
```

Properties

keyName

Specifies the name of the key that caused the error.

errorCode

Specifies the type of illegal key access that occurred

[7.4.3 - KeyManagement.CertificateChangeEvent](#)

This event indicates that the certificate module state has changed from Primary to Secondary.

| Payload | Type | Required |
|--|--------|----------|
| { "certificateChange": "secondary" } | string | |

Properties

certificateChange

Specifies change of the certificate state inside of the KeyManagement.

8 - Crypto Interface

This chapter defines the Crypto interface functionality and messages.

8.1 - Summary

TODO

8.2 - General Information

8.2.1 - DUKPT

Definitions and Abbreviations

| | |
|-------|------------------------------------|
| DUKPT | Derived Unique Key Per Transaction |
| BDK | Base Derivation Key |
| IPEK | Initial PIN Encryption Key |
| KSN | Key Serial Number. |
| TRSM | Tamper Resistant Security Module. |

For additional information see reference 45.

2.1 Default Key Name

The dukpt IPEK key is given a fixed name so multi-vendor clients can be developed without the need for vendor specific configuration tools.

If dukpt is supported, this key must be included in the KeyDetail output.

| Item Name | Description |
|-------------|--|
| "dukptIpek" | This key represents the IPEK, the derived future keys stored during import of the IPEK and the variant per transaction keys (PIN and optionally data and MAC). |

8.3 - Command Messages

8.3.1 - Crypto.GenerateRandom

This command is used to generate a random number.

Command Message

| Payload | Type | Required |
|------------------------------------|------|----------|
| { "timeout": 5000 integer } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "modeNotSupported", | string | |
| "randomNumber": Add example to YAML | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- modeNotSupported - The mode specified by modeOfUse is not supported.
- accessDenied - The encryption module is either not initialized or not ready for any vendor specific reason.

randomNumber

The generated random number.

Event Messages

None

[8.3.2 - Crypto.CryptoData](#)

The input data is either encrypted or decrypted using the specified or selected encryption mode. The input data is padded to the necessary length mandated by the encryption algorithm using the padding parameter. This input data is padded to necessary length mandated by the signature algorithm using padding parameter. Clients can use an alternative padding method by pre-formatting the data passed and combining this with the standard padding method. The start value (or Initialization Vector) can be provided as input data to this command, or it can be imported via TR-31 prior to requesting this command and referenced by name. The start value and start value key are both optional parameters.

Command Message

| Payload | Type | Required |
|------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |

```

"key": Add example to YAML,           string
"startValueKey": Add example to YAML, string
"startValue": Add example to YAML,    string
"padding": 0,                      integer
"cryptData": Add example to YAML,   string
"cryptoAttributes": {
"algorithm": "A",                 string
"modeOfUse": "D",                 string
"cryptoMethod": "ecb"             string
}
}

```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

key

Specifies the name of the stored key.

startValueKey

If startValue specifies an Initialization Vector (IV), then this parameter specifies the name of the stored key used to decrypt the startValue to obtain the IV. If startValue is not set and this parameter is set, then this parameter specifies the name of the IV that has been previously imported via TR-31. If this parameter is not set, startValue is used as the Initialization Vector.

startValue

The initialization vector for CBC / CFB encryption. If this parameter and startValueKey are both not set the default value for CBC / CFB is all zeroes.

padding

Specifies the padding character. The padding character is a full byte, e.g. 0xFF. The valid range is 0x00 to 0xFF.

cryptData

The data to be encrypted or decrypted formatted in Base64.

cryptoAttributes

This parameter specifies the encryption algorithm, cryptographic method, and mode to be used for this command. For a list of valid values see the [Capability.Attributes](#) field. The values specified must be compatible with the key identified by Key.

cryptoAttributes/algorithm

Specifies the encryption algorithms supported by [Crypto.CryptoData](#) command. The following values are possible:

- A - AES.
- D - DEA.
- R - RSA.
- T - Triple DEA (also referred to as TDEA).

cryptoAttributes/modeOfUse

Specifies the encryption mode supported by [Crypto.CryptoData](#) command. The following values are possible:

- D - Decrypt
- E - Encrypt

cryptoAttributes/cryptoMethod

Specifies the cryptographic method supported by the [Crypto.CryptoData](#) command. For symmetric encryption methods (keyUsage is 'D0'), this can be one of the following values:

- ecb - The ECB encryption method.
- cbc - The CBC encryption method.

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

- cfb - The CFB encryption method.
- ofb - The OFB encryption method.
- ctr - The CTR method defined in NIST SP800-38A.
- xts - The XTS method defined in NIST SP800-38E.

For asymmetric encryption methods (keyUsage is 'D1'), this can be one of the following values:

- rsaesPkcs1V15 - Use the RSAES_PKCS1-v1.5 algorithm.
- rsaesOaep - Use the RSAES OAEP algorithm.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "keyNotFound", | string | |
| "cryptData": Add example to YAML | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- keyNotFound - The specified key was not found.
- modeNotSupported - The mode specified by modeOfUse is not supported.
- accessDenied - The encryption module is either not initialized or not ready for any

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

vendor specific reason.

- keyNoValue - The specified key name was found but the corresponding key value has not been loaded.
- useViolation - The use specified by keyUsage is not supported.
- invalidKeyLength - The length of startValue is not supported or the length of an encryption key is not compatible with the encryption operation required.
- noChipTransactionActive - A chipcard key is used as encryption key and there is no chip transaction active.
- algorithmNotSupported - The algorithm specified by bAlgorithm is not supported.
- cryptoMethodNotSupported - The cryptographic method specified by cryptoMethod is not supported.

cryptData

The encrypted or decrypted data formatted in Base64.

Event Messages

- Pinpad.DUKPTKSNEvent

8.3.3 - Crypto.GenerateAuthentication

The Authentication data is generated using the specified mode. The available modes are defined in the [Crypto.Capabilities](#) command. This command can be used for Message Authentication Code generation (i.e. macing). The input data is padded to the necessary length mandated by the encryption algorithm using the padding parameter. This command can be used for asymmetric signature generation. This input data is padded to necessary length mandated by the signature algorithm using padding parameter. Clients can use an alternative padding method by pre-formatting the data passed and combining this with the standard padding method. The start value (or Initialization Vector) can be provided as input data to this command, or it can be imported via TR-31 prior to requesting this command and referenced by name. The start value and start value key are both optional parameters.

Command Message

| Payload | Type | Required |
|-----------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "key": Add example to YAML, | string | |

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

```

"startValueKey": Add example to YAML,      string
"startValue": Add example to YAML,         string
"padding": 0,                            integer
"compression": false,                   boolean
"authenticationData": Add example to YAML, string
"authenticationAttribute": {           object
    "modeOfUse": "G",                  string
    "cryptoMethod": "rsassaPkcs1V15",   string
    "hashAlgorithm": "sha1"            string
}
}

```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

key

Specifies the name of the stored key.

startValueKey

If startValue specifies an Initialization Vector (IV), then this property specifies the name of the stored key used to decrypt the startValue to obtain the IV. If startValue is not set and this property is also not set, then this property specifies the name of the IV that has been previously imported via TR-31. If this property is not set, startValue is used as the Initialization Vector.

startValue

The initialization vector for CBC / CFB encryption. If this parameter and startValueKey are both not set the default value for CBC / CFB is all zeroes.

padding

Specifies the padding character. The padding character is a full byte, e.g. 0xFF. The valid range is 0x00 to 0xFF.

compression

Specifies whether data is to be compressed (blanks removed) before building the mac. If compression is 0x00 no compression is selected, otherwise compression holds the representation of the blank character (e.g. 0x20 in ASCII or 0x40 in EBCDIC).

authenticationData

The data to be MACed, or signed formatted in base64.

authenticationAttribute

This parameter specifies the encryption algorithm, cryptographic method, and mode to be used for this command. For a list of valid values see the Attributes [Crypto.capability](#) field. The values specified must be compatible with the key identified by Key.

authenticationAttribute/modeOfUse

Specifies the encryption mode supported by the [Crypto.GenerateAuthentication](#) command. The following values are possible:

- G - Generate. This be used to generate a MAC.
- S - Signature

authenticationAttribute/cryptoMethod

Specifies the cryptographic method supported by the [Crypto.GenerateAuthentication](#) command. For asymmetric signature verification methods (keyUsage is 'S0', 'S1', or 'S2'), this can be one of the following values:

- rsassaPkcs1V15 - Use the RSASSA-PKCS1-v1.5 algorithm.
- rsassaPss - Use the RSASSA-PSS algorithm.

If keyUsage is specified as any of the MAC usages (i.e. 'M1'), then this proeprty should not be not set.

authenticationAttribute/hashAlgorithm

For asymmetric signature verification methods (keyUsage is 'S0', 'S1', or 'S2'), this can be one of the following values to be used. If keyUsage is specified as any of the MAC usages (i.e. 'M1'), then properties should not be set. this can be one of the following values:

- sha1 - The SHA 1 digest algorithm.
- sha256 - The SHA 256 digest algorithm, as defined in ISO/IEC 10118-3:2004 and FIPS 180-2.

Completion Message

| Payload | Type | Required |
|---|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "keyNotFound", | string | |
| "authenticationData": Add example to YAML | string | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- keyNotFound - The specified key was not found.
- modeNotSupported - The mode specified by modeOfUse is not supported.

- accessDenied - The encryption module is either not initialized or not ready for any vendor specific reason.
- keyNoValue - The specified key name was found but the corresponding key value has not been loaded.
- useViolation - The use specified by keyUsage is not supported.
- invalidKeyLength - The length of startValue is not supported or the length of an encryption key is not compatible with the encryption operation required.
- noChipTransactionActive - A chipcard key is used as encryption key and there is no chip transaction active.
- algorithmNotSupported - The algorithm specified by bAlgorithm is not supported.
- macInvalid - The MAC verification failed.
- signatureInvalid - The signature verification failed.
- cryptoMethodNotSupported - The cryptographic method specified by cryptoMethod is not supported.

authenticationData

The mac value or signature formatted in Base64.

Event Messages

- Pinpad.DUKPTKSNEvent

8.3.4 - Crypto.VerifyAuthentication

The authentication data is verified using the specified mode. The available modes are defined in the [Crypto.Capabilities](#) command. This command can be used for MAC and signature verification. The input data is padded to the necessary length mandated by the encryption algorithm using the padding parameter. This input data is padded to necessary length mandated by the signature algorithm using padding parameter. Clients can use an alternative padding method by pre-formatting the data passed and combining this with the standard padding method. The start value (or Initialization Vector) can be provided as input data to this command, or it can be imported via TR-31 prior to requesting this command and referenced by name. The start value and start value key are both optional parameters.

Command Message

| Payload | Type | Required |
|---------|------|----------|
| { | | |

| | |
|--|---------|
| " timeout<td>integer</td> | integer |
| " key<td>string</td> | string |
| " startValueKey<td>string</td> | string |
| " startValue<td>string</td> | string |
| " padding<td>integer</td> | integer |
| " compression<td>boolean</td> | boolean |
| " authenticationData<td>string</td> | string |
| " verifyData<td>string</td> | string |
| " verifyAttributes<td>object</td> | object |
| " modeOfUse<td>string</td> | string |
| " cryptoMethod<td>string</td> | string |
| " hashAlgorithm<td>string</td> | string |
| } | |
| } | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

key

Specifies the name of the stored key.

startValueKey

If startValue specifies an Initialization Vector (IV), then this property specifies the name of the stored key used to decrypt the startValue to obtain the IV. If startValue is not set and this property is also not set, then this property specifies the name of the IV that has been previously imported via TR-31. If this property is not set, startValue is used as the

Initialization Vector.

startValue

The initialization vector for CBC / CFB encryption. If this parameter and startValueKey are both not set the default value for CBC / CFB is all zeroes.

padding

Specifies the padding character. The padding character is a full byte, e.g. 0xFF. The valid range is 0x00 to 0xFF.

compression

Specifies whether data is to be compressed (blanks removed) before building the mac. If compression is 0x00 no compression is selected, otherwise compression holds the representation of the blank character (e.g. 0x20 in ASCII or 0x40 in EBCDIC).

authenticationData

The device will either generate a MAC or sign the authenticationData and compare with verifyData formatted in base64.

verifyData

The authentication data to be verified by MAC or signature formatted in base64.

verifyAttributes

This parameter specifies the encryption algorithm, cryptographic method, and mode to be used for this command. For a list of valid values see the Attributes [Crypto.capability](#) field. The values specified must be compatible with the key identified by Key.

verifyAttributes/modeOfUse

Specifies the encryption mode supported by [Crypto.VerifyAuthentication](#) command. The following values are possible:

- S - Signature.
- V - Verify. This be used to verify a MAC.

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

verifyAttributes/cryptoMethod

Specifies the cryptographic method supported by the [Crypto.VerifyAuthentication](#) command. For asymmetric signature verification methods (bKeyUsage is 'S0', 'S1', or 'S2'), this can be one of the following values.

- rsassaPkcs1V15 - Use the RSASSA-PKCS1-v1.5 algorithm.
- rsassaPss - Use the RSASSA-PSS algorithm.

If keyUsage is specified as any of the MAC usages (i.e. 'M1'), then this property should not be set.

verifyAttributes/hashAlgorithm

For asymmetric signature verification methods (keyUsage is 'S0', 'S1', or 'S2'), this can be one of the following values to be used. If keyUsage is specified as any of the MAC usages (i.e. 'M1'), then properties should not be set or both 'sha1' and 'sha256' are false. **sha1**: The SHA 1 digest algorithm.

- sha256 - The SHA 256 digest algorithm, as defined in ISO/IEC 10118-3:2004 and FIPS 180-2.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "keyNotFound" | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional

information

errorCode

Specifies the error code if applicable. The following values are possible:

- keyNotFound - The specified key was not found.
- modeNotSupported - The mode specified by modeOfUse is not supported.
- accessDenied - The encryption module is either not initialized or not ready for any vendor specific reason.
- keyNoValue - The specified key name was found but the corresponding key value has not been loaded.
- useViolation - The use specified by keyUsage is not supported.
- invalidKeyLength - The length of startValue is not supported or the length of an encryption key is not compatible with the encryption operation required.
- noChipTransactionActive - A chipcard key is used as encryption key and there is no chip transaction active.
- algorithmNotSupported - The algorithm specified by bAlgorithm is not supported.
- macInvalid - The MAC verification failed.
- signatureInvalid - The signature verification failed.
- cryptoMethodNotSupported - The cryptographic method specified by cryptoMethod is not supported.

Event Messages

- [Pinpad.DUKPTKSNEvent](#)

8.3.5 - Crypto.Digest

This command is used to compute a hash code on a stream of data using the specified hash algorithm. This command can be used to verify emv static and dynamic data.

Command Message

| Payload | Type | Required |
|------------------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "hashAlgorithm": "sha1", | string | |
| "digestInput": Add example to YAML | string | |

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

}

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

hashAlgorithm

Specifies which hash algorithm should be used to calculate the hash. See the [Crypto.Capabilities](#) section for valid algorithms.

digestInput

Contains the length and the data to be hashed formatted in base64.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "accessDenied", | string | |
| "digestOutput": Add example to YAML | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- accessDenied - The encryption module is either not initialized or not ready for any vendor specific reason.

digestOutput

Contains the length and the data containing the calculated has.

Event Messages

None

8.4 - Event Messages

8.4.1 - Pinpad.DUKPTKSNEvent

This event sends the DUKPT KSN of the key used in the command. The receiving TRSM uses this to derive the key from the BDK.

| Payload | Type | Required |
|-----------------------------|--------|----------|
| { | | |
| "key": Add example to YAML, | string | |
| "ksn": Add example to YAML | string | |
| } | | |

Properties

key

Specifies the name of the DUKPT Key derivation key.

ksn

structure that contains the KSN formatted in base64.

8.5 - Unsolicited Messages

8.5.1 - [Crypto.IllegalKeyAccessEvent](#)

This event specifies that an error occurred accessing an encryption key. Possible situations for generating this event are listed in the description of lErrorCode.

| Payload | Type | Required |
|---------------------------------|--------|----------|
| { | | |
| "keyName": Add example to YAML, | string | |
| "errorCode": "keynotfound" | string | |
| } | | |

Properties

keyName

Specifies the name of the key that caused the error.

errorCode

Specifies the type of illegal key access that occurred

9 - Keyboard Interface

This chapter defines the Keyboard interface functionality and messages.

9.1 - Summary

This section describes the general interface for the following functions:

- Entering Personal Identification Numbers (PINs)
- Clear text data handling
- Function key handling

If the PIN pad device has local display capability, display handling should be handled using the Text Terminal Unit (TTU) interface. The adoption of this specification does not imply the adoption of a specific security standard.

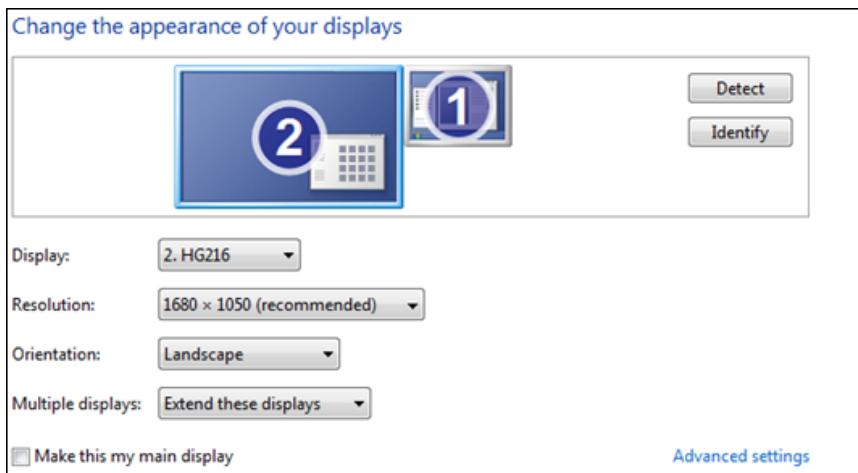
- Only numeric PIN pads are handled in this specification.

9.2 - General Information

9.2.1 - Encrypting Touch Screen (ETS)

An encrypting touch screen device is a touch screen securely attached to a cryptographic device. It can be used as an alternative to an encrypting pin pad (EPP). It supports key management, encryption and decryption.

It is assumed that the ETS is a combined device. It overlays a display monitor which is used to display lead-through for a transaction. It is assumed that the display monitor is part of the Windows desktop, and can be the Windows primary monitor or any other monitor on the desktop. E.g. the following diagram shows 2 monitors extended across the desktop, with monitor 1 being the primary monitor and the ETS being overlaid on monitor 2 whose origin is (-1680,0).



The touch screen can optionally be used as a “mouse” for client purposes, while PIN operations are not in progress or optionally when non-secure PIN commands are in progress.

The CEN interface supports two types of ETS

- Those which activate touch areas defined by the client.
- Those which activate a random variation of touch areas defined by the client.

The Service Provider, when reporting its capabilities, reports the absolute position of the ETS in Windows desktop coordinates. This allows the client to locate the ETS device in a multi-monitor system and relate it to a monitor on the desktop.

At any point in time, a single touch area of the ETS can operate in one of 4 modes:-

- **Mouse mode** - a "touch" simulates a mouse click. This mode is optional. This may not be supported by some ETS devices. Configuration of the click is vendor specific. e.g. WM_LBUTTONDOWN. This is also the mode that, if supported, is active when none of the other modes are active.
- **Data mode** - a "touch" maps to an key and the value of the key is returned in an event (as in clear numeric entry using [Keyboard.DataEntry](#)).
- **PIN mode** - a "touch" maps to an key and the value of the key is returned in an event only if the key pressed is not Fk0 through Fk9 (as in PIN entry using [Keyboard.PinEntry](#)).
- **Secure mode** - a "touch" maps to an key and the value of the key is returned in an event only if the key pressed is not Fk0 through Fk9 and not Fk_A through Fk_F (as in key entry using [Keyboard.SecureKeyEntry](#)).

The following concepts are introduced to define the relationship between the monitor and the ETS:-

- **Touch Key** – an area of the monitor which reacts to touch in Data, PIN and Secure modes.
- **Touch Frame** – an area of the monitor onto which Touch Keys can be placed. There can be one or more Touch Frames. There may be just one Touch Frame which covers the whole monitor. Areas within a Touch Frame, not defined as a Touch Key, do not react to touch. Generally in PIN and Secure modes, there would be only one Touch Frame covering the whole monitor. An empty Touch Frame disables that part of the monitor.
- **Mouse area** – an area outside of all Touch Frames in which touches behave like a mouse.
- Thus Data, PIN and Secure modes operate in a single Touch Frame or multiple Touch Frames. Mouse mode operates outside a Touch Frame, and is optional.

Note that there is a perceived risk in separating the drawing functionality from the touch functionality, but this type of risk is present in today's keyboard based systems. e.g. a client can draw on a monitor to prompt the user to enter a PIN and then enables the EPP for clear data entry. So the risk is no different than with an EPP – the client has to be trusted.

Depending upon the type of device, the client must then either inform the Service Provider as to the active key positions in the form of Touch Frames and Touch Keys using the [Keyboard.DefineLayout](#) command, or obtain them from the Service Provider using the [Keyboard.GetLayout](#) command. This collection is now referred to as a "Touch Keyboard definition".

The client then uses the normal PIN commands to enable the touch keyboard definition on the ETS device:

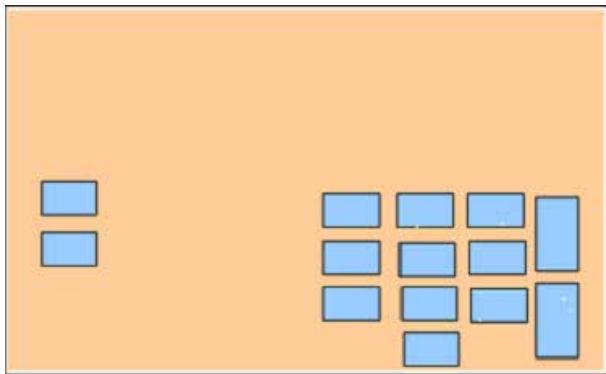
- PIN entry [Keyboard.PinEntry](#)
- Clear data entry [Keyboard.DataEntry](#)
- Secure key entry [Keyboard.SecureKeyEntry](#)

These commands are referred to as "keyboard entry commands" throughout the remainder of this document.

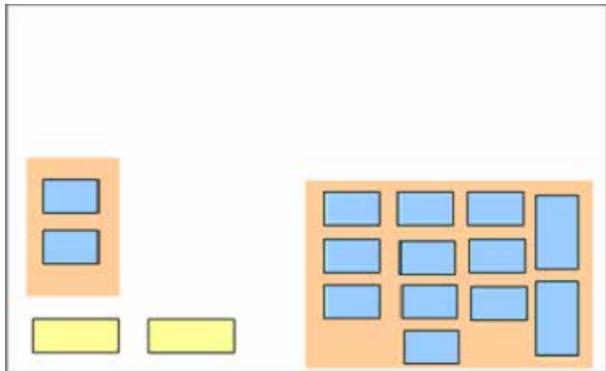
PCI compliance means that [Keyboard.PinEntry](#) and [Keyboard.SecureKeyEntry](#) can only be used with a single Touch Frame that covers the entire monitor. i.e. Mouse mode cannot be mixed with either PIN or Secure mode. If a Touch Key (or areas) is defined for an key value and that key value is not subsequently specified as active in a [Keyboard.PinEntry](#), [Keyboard.DataEntry](#) or [Keyboard.SecureKeyEntry](#) command, then the Touch Key is made inactive.

Layouts defined with the [Keyboard.DefineLayout](#) command are persistent.

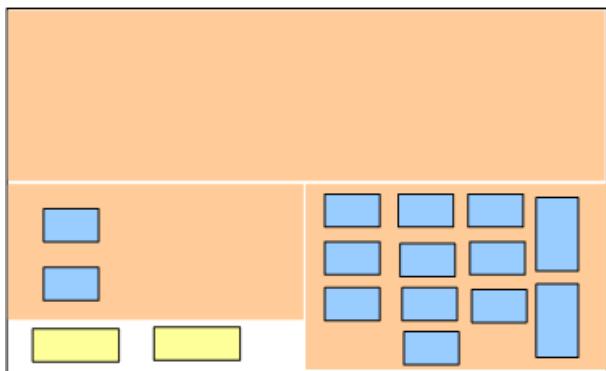
Example 1 – this screen only uses Data mode – the entire screen is a Touch Frame. Mouse mode is not used.



Example 2 – this shows a monitor with two Touch Frames and 14 Touch Keys. The space within the Touch Frames not defined by a Touch Key are inactive (do not respond to touch). All areas outside a Touch Frame operate in Mouse mode. This example shows two Mouse mode "keys". e.g. Windows "Button", HTML "BUTTON" or a custom control. Other touches in Mouse mode are normally dealt with by the client event engine. However, this can be restricted – see example 3.



Example 3 - this screen uses Mouse and Data modes – Mouse mode is used only in a restricted area. The touch keyboard definition has 3 frames. Frame 1 has no Touch Keys. Frame 2 has 2 Touch Keys; Frame 3 has 12 Touch Keys.



9.2.2 - Secure Key Entry

This section provides additional information to describe how encryption keys are entered securely through the PIN pad keyboard and also provides examples of possible keyboard layouts.

Keyboard Layout

The following sections describe what is returned within the GetSecureKeyDetail output parameters to describe the physical keyboard layout. These descriptions are purely examples to help understand the usage of the parameters they do not indicate a specific layout per Key Entry Mode.

In the following section all references to parameters relate to the output fields of the GetSecureKeyDetail command.

When keyEntryMode represents a regular shaped PIN pad (regUnique or regShift) then hexKeys must contain one entry for each physical key on the PIN pad (i.e. the product of Rows by Columns). On a regular shaped PIN pad the client can choose to ignore the position and size data and just use the rows and columns parameters to define the layout. However, a Service Provider must return the position and size data for each key.

keyEntryMode == regUnique

When keyEntryMode is regUnique then the values in the array report which physical keys are associated with the function keys 0-9, A-F and any other function keys that can be enabled as defined in the funcKeyDetail parameter. Any positions on the PIN pad that are not used must be defined as a fkUnused in the fk and shiftFk field of the hexKeys structure.

| | | | |
|-----|---|-----|------------|
| 1 | 2 | 3 | Clear (A) |
| 4 | 5 | 6 | Cancel (B) |
| 7 | 8 | 9 | Enter (C) |
| (D) | 0 | (E) | (F) |

In the above example, where all keys are the same size and the hex digits are located as shown the hexKeys will contain the entries in the array as defined in the following table.

| Index | xPos | yPos | xSize | ySize | fk | shiftfk |
|-------|------|------|-------|-------|-----|----------|
| 0 | 0 | 0 | 250 | 250 | fk1 | fkUnused |
| 1 | 250 | 0 | 250 | 250 | fk2 | fkUnused |
| 2 | 500 | 0 | 250 | 250 | fk3 | fkUnused |
| 3 | 750 | 0 | 250 | 250 | fkA | fkUnused |
| 4 | 0 | 250 | 250 | 250 | fk4 | fkUnused |
| 5 | 250 | 250 | 250 | 250 | fk5 | fkUnused |
| 6 | 500 | 250 | 250 | 250 | fk6 | fkUnused |

| | | | | | | |
|----|-----|-----|-----|-----|-----|----------|
| 7 | 750 | 250 | 250 | 250 | fkB | fkUnused |
| 8 | 0 | 500 | 250 | 250 | fk7 | fkUnused |
| 9 | 250 | 500 | 250 | 250 | fk8 | fkUnused |
| 10 | 500 | 500 | 250 | 250 | fk9 | fkUnused |
| 11 | 750 | 500 | 250 | 250 | fkC | fkUnused |
| 12 | 0 | 750 | 250 | 250 | fkD | fkUnused |
| 13 | 250 | 750 | 250 | 250 | fk0 | fkUnused |
| 14 | 500 | 750 | 250 | 250 | fkE | fkUnused |
| 15 | 750 | 750 | 250 | 250 | fkF | fkUnused |

keyEntryMode == regShift

When keyEntryMode is regShift then the values in the array report which physical keys are associated with the function keys 0-9, A-F, and the shift key as defined in the funcKeyDetail parameter. Other function keys as defined by the funcKeyDetail parameter that can be enabled must also be reported. Any positions on the PIN pad that are not used must be defined as a fkUnused in the fk and shiftFk field of the hexKeys structure. Digits 0 to 9 are accessed through the numeric keys as usual. Digits A to F are accessed by using the shift key in combination with another function key, e.g. shift-0 (zero) is hex digit A.

| | | | |
|-------|-------|-------|--------|
| 1 (B) | 2 (C) | 3 (D) | Clear |
| 4 (E) | 5 (F) | 6 | Cancel |
| 7 | 8 | 9 | Enter |
| SHIFT | 0 (A) | | |

In the above example, where all keys are the same size and the hex digits 'A' to 'F' are accessed through shift '0' to '5', then the hexKeys will contain the entries in the array as defined in the following table.

| Index | xPos | yPos | xSize | ySize | fk | shiftfk |
|-------|------|------|-------|-------|----------|----------|
| 0 | 0 | 0 | 250 | 250 | fk1 | fkB |
| 1 | 250 | 0 | 250 | 250 | fk2 | fkC |
| 2 | 500 | 0 | 250 | 250 | fk3 | fkD |
| 3 | 750 | 0 | 250 | 250 | fkClear | fkUnused |
| 4 | 0 | 250 | 250 | 250 | fk4 | fkE |
| 5 | 250 | 250 | 250 | 250 | fk5 | fkF |
| 6 | 500 | 250 | 250 | 250 | fk6 | fkUnused |
| 7 | 750 | 250 | 250 | 250 | fkCancel | fkUnused |
| 8 | 0 | 500 | 250 | 250 | fk7 | fkUnused |
| 9 | 250 | 500 | 250 | 250 | fk8 | fkUnused |
| 10 | 500 | 500 | 250 | 250 | fk9 | fkUnused |

| | | | | | | |
|----|-----|-----|-----|-----|----------|----------|
| 11 | 750 | 500 | 250 | 250 | fkEnter | fkUnused |
| 12 | 0 | 750 | 250 | 250 | fkShift | fkUnused |
| 13 | 250 | 750 | 250 | 250 | fk0 | fkA |
| 14 | 500 | 750 | 250 | 250 | fkUnused | fkUnused |
| 15 | 750 | 750 | 250 | 250 | fkUnused | fkUnused |

keyEntryMode == irregShift

When keyEntryMode represents an irregular shaped PIN pad the rows and columns parameters define the ratio of the width to height, i.e. square if the parameters are the same or rectangular if Columns is larger than rows, etc. A Service Provider must return the position and size data for each key reported.

When keyEntryMode is irregShift then the values in the array must be the function keys codes for 0-9 and the shift key as defined in the funcKeyDetail parameter. Other function keys as defined by the funcKeyDetail parameter that can be enabled must also be reported. Any positions on the PIN pad that are not used must be defined as a fkUnused in the fk and shiftfk field of the hexKeys structure. Digits 0 to 9 are accessed through the numeric keys as usual. Digits A - F are accessed by using the shift key in combination with another function key, e.g. shift-0(zero) is hex digit A.

| | | | |
|-------|-------|-------|--------|
| 1 (B) | 2 (C) | 3 (D) | Clear |
| 4 (E) | 5 (F) | 6 | Cancel |
| 7 | 8 | 9 | Enter |
| 0 (A) | SHIFT | | |

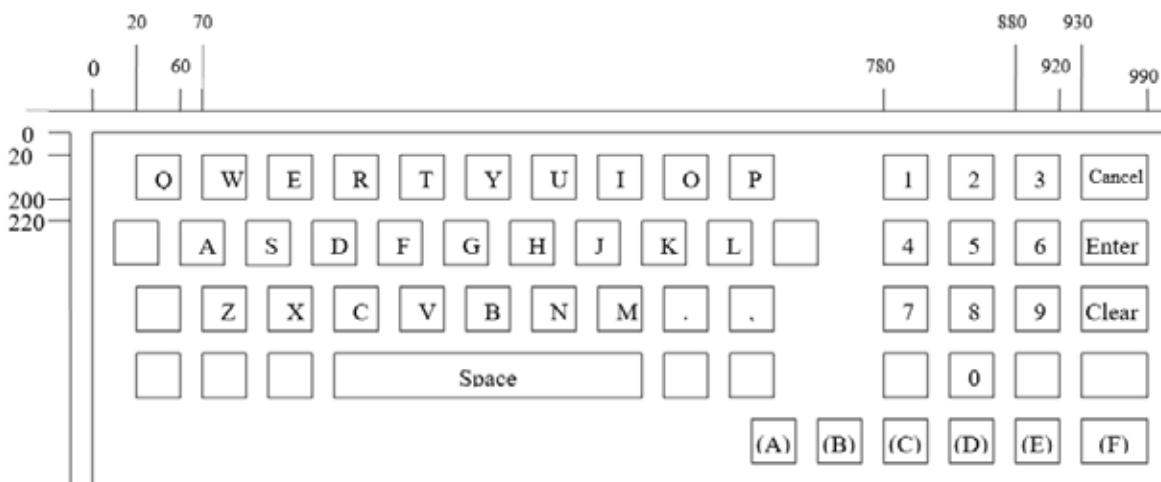
In the above example, where the hex digits 'A' to 'F' are accessed through shift '0' to '5', columns will be 4, rows will be 5 and the hexKeys will contain the entries in the array as defined in the following table.

| Index | xPos | yPos | xSize | ySize | fk | shiftfk |
|-------|------|------|-------|-------|----------|----------|
| 0 | 0 | 0 | 250 | 200 | fk1 | fkB |
| 1 | 250 | 0 | 250 | 200 | fk2 | fkC |
| 2 | 500 | 0 | 250 | 200 | fk3 | fkD |
| 3 | 750 | 0 | 250 | 200 | fkClear | fkUnused |
| 4 | 0 | 200 | 250 | 200 | fk4 | fkE |
| 5 | 250 | 200 | 250 | 200 | fk5 | fkF |
| 6 | 500 | 200 | 250 | 200 | fk6 | fkUnused |
| 7 | 750 | 200 | 250 | 200 | fkCancel | fkUnused |
| 8 | 0 | 400 | 250 | 200 | fk7 | fkUnused |
| 9 | 250 | 400 | 250 | 200 | fk8 | fkUnused |

| | | | | | | |
|----|-----|-----|------|-----|----------|----------|
| 10 | 500 | 400 | 250 | 200 | fk9 | fkUnused |
| 11 | 750 | 400 | 250 | 200 | fkEnter | fkUnused |
| 12 | 0 | 600 | 250 | 200 | fkUnused | fkUnused |
| 13 | 250 | 600 | 250 | 200 | fk0 | fkA |
| 14 | 500 | 600 | 250 | 200 | fkUnused | fkUnused |
| 15 | 750 | 600 | 250 | 200 | fkUnused | fkUnused |
| 16 | 0 | 800 | 1000 | 200 | fkShift | fkUnused |

keyEntryMode == irregUnique

When keyEntryMode is irregUnique then the values in the array report which physical keys are associated with the function keys 0-9, A-F and any other function keys that can be enabled as defined in the FuncKeyDetail parameter. The rows and columns parameters define the ratio of the width to height, i.e. square if the parameters are the same or rectangular if columns is larger than rows, etc. A Service Provider must return the position and size data for each key.



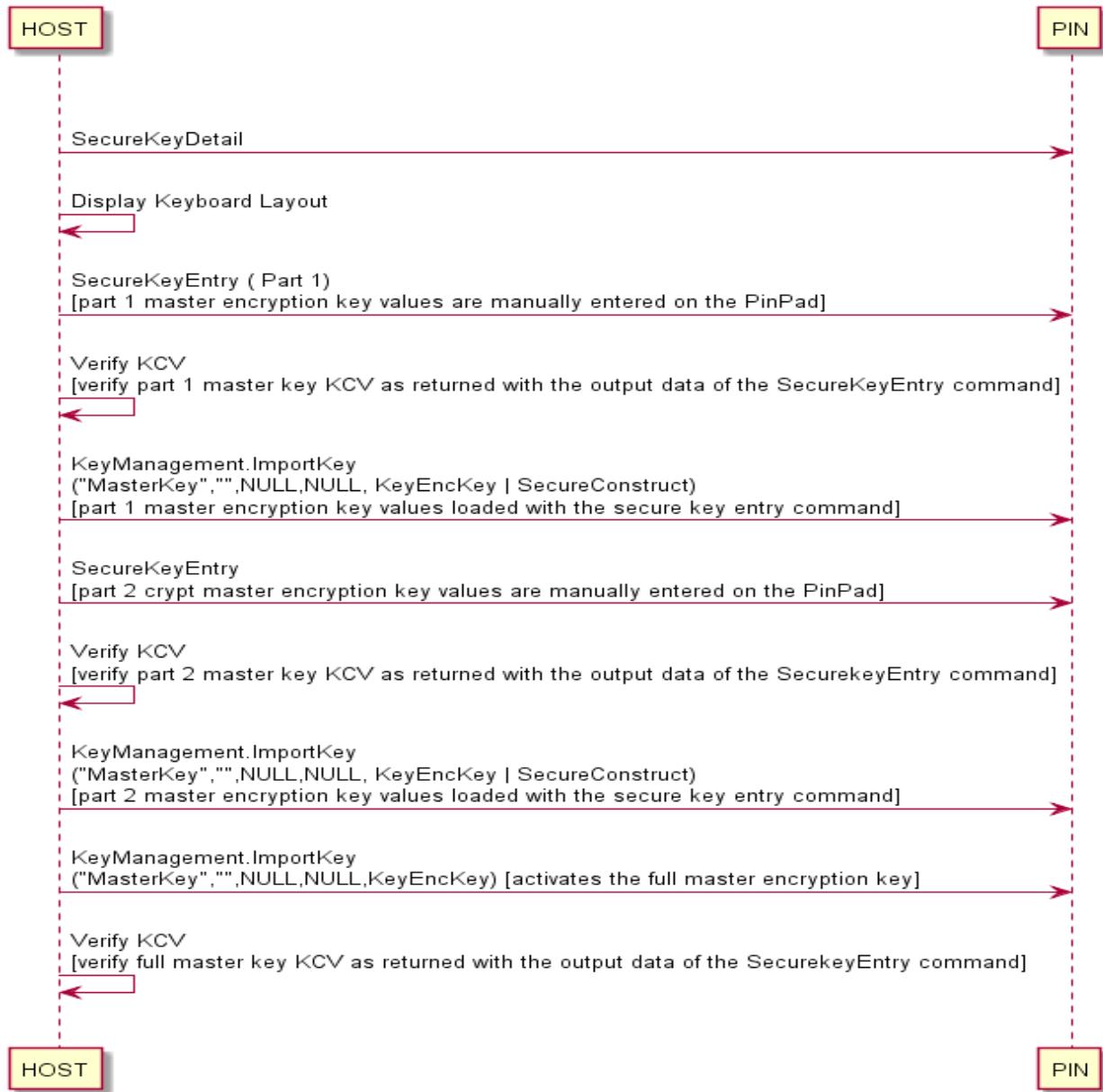
In the above example, where an alphanumeric keyboard supports secure key entry and the hex digits are located as shown, the hexKeys will contain the entries in the array as defined in the following table. All the hex digits and function keys that can be enabled must be included in the array; in addition any keys that would help a client display an image of the keyboard can be included. In this example only the PIN pad digits (the keys on the right) and the unique hex digits are reported. Note that the position data in this example may not be 100% accurate as the diagram is not to scale.

| Index | xPos | yPos | xSize | ySize | fk | shiftfk |
|-------|------|------|-------|-------|-----|----------|
| 0 | 780 | 18 | 40 | 180 | fk1 | fkUnused |
| 1 | 830 | 18 | 40 | 180 | fk2 | fkUnused |

| | | | | | | |
|----|-----|-----|----|-----|----------|----------|
| 2 | 880 | 18 | 40 | 180 | fk3 | fkUnused |
| 3 | 930 | 18 | 60 | 180 | fkCancel | fkUnused |
| 4 | 780 | 216 | 40 | 180 | fk4 | fkUnused |
| 5 | 830 | 216 | 40 | 180 | fk5 | fkUnused |
| 6 | 880 | 216 | 40 | 180 | fk6 | fkUnused |
| 7 | 930 | 216 | 60 | 180 | fkEnter | fkUnused |
| 8 | 780 | 414 | 40 | 180 | fk7 | fkUnused |
| 9 | 830 | 414 | 40 | 180 | fk8 | fkUnused |
| 10 | 880 | 414 | 40 | 180 | fk9 | fkUnused |
| 11 | 930 | 414 | 60 | 180 | fkClear | fkUnused |
| 12 | 780 | 612 | 40 | 180 | fkUnused | fkUnused |
| 13 | 830 | 612 | 40 | 180 | fk0 | fkUnused |
| 14 | 880 | 612 | 40 | 180 | fkUnused | fkUnused |
| 15 | 930 | 612 | 60 | 180 | fkUnused | fkUnused |
| 16 | 680 | 810 | 40 | 180 | fkA | fkUnused |
| 17 | 730 | 810 | 40 | 180 | fkB | fkUnused |
| 18 | 780 | 810 | 40 | 180 | fkC | fkUnused |
| 19 | 830 | 810 | 40 | 180 | fkD | fkUnused |
| 20 | 880 | 810 | 40 | 180 | fkE | fkUnused |
| 21 | 930 | 810 | 60 | 180 | fkF | fkUnused |

9.2.3 - Command Usage

This section provides an example of the sequence of commands required to enter an encryption key securely. In the following sequence, the client retrieves the keyboard secure key entry mode and associated keyboard layout and displays an image of the keyboard for the user. It then gets the first key part, verifies the KCV for the key part and stores it. The sequence is repeated for the second key part and then finally the key part is activated.



9.3 - Command Messages

9.3.1 - Keyboard.GetFuncKeyDetail

This command returns information about the names of the Function Keys supported by the device. Location information is also returned for the supported FDKs (Function Descriptor Keys). This includes screen overlay FDKs.

This command should be issued before the first call to [Keyboard.PinEntry](#) or [Keyboard.DataEntry](#) to determine which Function Keys (FKs) and Function Descriptor

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

Keys (FDKs) are available and where the FDKs are located. Then, in these two commands, they can then be specified as Active and Terminate keys and options on the customer screen can be aligned with the active FDKs.

Command Message

| Payload | Type | Required |
|---------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "fdkMask": "functionKeys" | string | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

fdkMask

Mask for the fdks for which additional information is requested.

Completion Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "funcMask": { | object | |
| "fk0": false, | boolean | |
| "fk1": false, | boolean | |

| | |
|-----------------------|---------|
| "fk2": false, | boolean |
| "fk3": false, | boolean |
| "fk4": false, | boolean |
| "fk5": false, | boolean |
| "fk6": false, | boolean |
| "fk7": false, | boolean |
| "fk8": false, | boolean |
| "fk9": false, | boolean |
| "fkA": false, | boolean |
| "fkB": false, | boolean |
| "fkC": false, | boolean |
| "fkD": false, | boolean |
| "fkE": false, | boolean |
| "fkF": false, | boolean |
| "fkEnter": false, | boolean |
| "fkCancel": false, | boolean |
| "fkClear": false, | boolean |
| "fkBackspace": false, | boolean |
| "fkHelp": false, | boolean |
| "fkDecPoint": false, | boolean |
| "fk00": false, | boolean |
| "fk000": false, | boolean |
| "fkShift": false, | boolean |
| "fkRES01": false, | boolean |
| "fkRES02": false, | boolean |
| "fkRES03": false, | boolean |

```

"fkRES04": false,           boolean
"fkRES05": false,           boolean
"fkRES06": false,           boolean
"fkRES07": false,           boolean
"fkRES08": false,           boolean
"fkOEM01": false,           boolean
"fkOEM02": false,           boolean
"fkOEM03": false,           boolean
"fkOEM04": false,           boolean
"fkOEM05": false,           boolean
"fkOEM06": false            boolean
},
"fdks": [{                  array (object)
  "fdk": "fk_fdk01",        string
  "xPosition": 0,           integer
  "yPosition": 0             integer
}]
}

```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

funcMask

Specifies the function keys available for this physical device.

funcMask/fk0

default: false

funcMask/fk1

default: false

funcMask/fk2

default: false

funcMask/fk3

default: false

funcMask/fk4

default: false

funcMask/fk5

default: false

funcMask/fk6

default: false

funcMask/fk7

default: false

funcMask/fk8

default: false

funcMask/fk9

default: false

funcMask/fkA

default: false

funcMask/fkB

default: false

funcMask/fkC

default: false

funcMask/fkD

default: false

funcMask/fkE

default: false

funcMask/fkF

default: false

funcMask/fkEnter

default: false

funcMask/fkCancel

default: false

funcMask/fkClear

default: false

funcMask/fkBackspace

default: false

funcMask/fkHelp

default: false

funcMask/fkDecPoint

default: false

funcMask/fk00

default: false

funcMask/fk000

default: false

funcMask/fkShift

default: false

funcMask/fkRES01

default: false

funcMask/fkRES02

default: false

funcMask/fkRES03

default: false

funcMask/fkRES04

default: false

funcMask/fkRES05

default: false

funcMask/fkRES06

default: false

funcMask/fkRES07

default: false

funcMask/fkRES08

default: false

funcMask/fkOEM01

default: false

funcMask/fkOEM02

default: false

funcMask/fkOEM03

default: false

funcMask/fkOEM04

default: false

funcMask/fkOEM05

default: false

funcMask/fkOEM06

default: false

fdks

It is the responsibility of the client to identify the mapping between the FDK code and the physical location of the FDK. An empty array if no FDKs are requested or supported.

fdks/fdk

Specifies the code returned by this FDK, defined as one of the following values:

fdks/xPosition

For FDKs, specifies the screen position the FDK relates to. This position is relative to the top of the screen expressed as a percentage of the height of the screen. For FDKs above or below the screen this will be 0 (above) or 100 (below).

fdks/yPosition

For FDKs, specifies the screen position the FDK relates to. This position is relative to the Left Hand side of the screen expressed as a percentage of the width of the screen. For FDKs along the side of the screen this will be 0 (left side) or 100 (right side, user's view).

Event Messages

None

[9.3.2 - Keyboard.GetLayout](#)

This command allows a client to retrieve layout information for any device. Either one layout or all defined layouts can be retrieved with a single request of this command.

There can be a layout for each of the different types of keyboard entry modes, if the vendor and the hardware support these different methods. The types of keyboard entry modes are: (1) Data Entry mode which corresponds to the [Keyboard.DataEntry](#) command, (2) PIN Entry mode which corresponds to the [Keyboard.PinEntry](#) command, (3) Secure Key Entry mode which corresponds to the [Keyboard.SecureKeyEntry](#) command. The layouts can be

preloaded into the device, if the device supports this, or a single layout can be loaded into the device immediately prior to the keyboard command being requested.

Command Message

| Payload | Type | Required |
|---------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "entryMode": "data" | string | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

entryMode

Specifies entry mode to be returned

Completion Message

| Payload | Type | Required |
|----------------------------------|---------|----------|
| { | | |
| "errorCode": "modeNotSupported", | string | |
| "entryMode": { | object | |
| "data": false, | boolean | |
| "pin": false, | boolean | |
| "secure": false | boolean | |
| , | | |

```

"frames": [{                                array (object)
    "xPos": 0,                            integer
    "yPos": 0,                            integer
    "xSize": 0,                           integer
    "ySize": 0,                           integer
    "floatAction": {                      object
        "floatX": false,                  boolean
        "floatY": false                  boolean
    },
    "fks": [{                                array (object)
        "xPos": 0,                          integer
        "yPos": 0,                          integer
        "xSize": 0,                         integer
        "ySize": 0,                         integer
        "fk": "fk0",                       string
        "shiftFK": "fk0"                   string
    }]
}
}

```

Properties

errorCode

Specifies the error code if applicable. The following values are possible:

- modeNotSupported - The specified entry mode is not supported.

entryMode

Specifies entry mode to be returned. It can be one of the following flags, or zero to return all supported entry modes.

entryMode/data

Specifies that the layout be applied to the [Keyboard.DataEntry](#) method.

default: false

entryMode/pin

Specifies that the layout be applied to the [Keyboard.PinEntry](#) method.

default: false

entryMode/secure

Specifies that the layout be applied to the [Keyboard.SecurekeyEntry](#) method.

default: false

frames

There can be one or more frame structures included

frames/xPos

For ETS, specifies the left coordinate of the frame as an offset from the left edge of the screen. For all other device types, this value is ignored

frames/yPos

For ETS, specifies the top coordinate of the frame as an offset from the top edge of the screen. For all other device types, this value is ignored

frames/xSize

For ETS, specifies the width of the frame. For all other device types, this value is ignored

frames/ySize

For ETS, specifies the height of the frame. For all other device types, this value is ignored

frames/floatAction

Specifies if the device can float the touch keyboards

frames/floatAction/floatX

Specifies that the PIN device will randomly shift the layout in a horizontal direction

default: false

frames/floatAction/floatY

Specifies that the PIN device will randomly shift the layout in a vertical direction

default: false

frames/fks

Defining details of the keys in the keyboard.

frames/fks/xPos

Specifies the position of the top left corner of the FK relative to the left hand side of the layout. For ETS devices, must be in the range defined in the frame. For non-ETS devices, must be a value between 0 and 999, where 0 is the left edge and 999 is the right edge.

frames/fks/yPos

Specifies the position of the top left corner of the FK relative to the left hand side of the layout. For ETS devices, must be in the range defined in the frame. For non-ETS devices, must be a value between 0 and 999, where 0 is the top edge and 999 is the bottom edge.

frames/fks/xSize

Specifies the FK width. For ETS, width is measured in pixels. For non-ETS devices, width is expressed as a value between 1 and 1000, where 1 is the smallest possible size and 1000 is the full width of the layout.

frames/fks/ySize

Specifies the FK height. For ETS, height is measured in pixels. For non-ETS devices, height is expressed as a value between 1 and 1000, where 1 is the smallest possible size and 1000 is

the full height of the layout.

frames/fks/fk

Specifies the FK code associated with the physical area in non-shifted mode.

frames/fks/shiftFK

Specifies the FK code associated with the physical key in shifted mode.

Event Messages

None

[9.3.3 - Keyboard.PinEntry](#)

This function stores the pin entry via the pin pad device. From the point this function is invoked, pin digit entries are not passed to the client. For each pin digit, or any other active key entered, an execute notification event [Keyboard.KeyEvent](#) is sent in order to allow a client to perform the appropriate display action (i.e. when the pin pad has no integrated display). The client is not informed of the value entered. The execute notification only informs that a key has been depressed.

The [Keyboard.EnterDataEvent](#) will be generated when the PIN pad is ready for the user to start entering data.

Some PIN pad devices do not inform the client as each PIN digit is entered, but locally process the PIN entry based upon minimum pin length and maximum PIN length input parameters.

When the maximum number of pin digits is entered and the flag autoEnd is true, or a terminating key is pressed after the minimum number of pin digits is entered, the command completes. If the key is a terminator key and is pressed, then the command will complete successfully even if the minimum number of pin digits has not been entered.

Terminating FDKs can have the functionality of (terminates only if minimum length has been reached) or (can terminate before minimum length is reached). The configuration of this functionality is vendor specific.

If maxLen is zero, the Service Provider does not terminate the command unless the client sets terminateKeys or terminateFDKs. In the event that terminateKeys or terminateFDKs are not set and maxLen is zero, the command will not terminate and the client must issue a Cancel command.

If active the fkCancel and fkClear keys will cause the PIN buffer to be cleared. The fkBackspace key will cause the last key in the PIN buffer to be removed.

Terminating keys have to be active keys to operate.

If this command is cancelled by a CancelAsyncRequest the PIN buffer is not cleared.

If maxLen has been met and autoEnd is set to False, then all numeric keys will automatically be disabled. If the clear or backspace key is pressed to reduce the number of entered keys, the numeric keys will be re-enabled.

If the enter key (or FDK representing the enter key - note that the association of an FDK to enter functionality is vendor specific) is pressed prior to minLen being met, then the enter key or FDK is ignored. In some cases the PIN pad device cannot ignore the enter key then the command will complete normally. To handle these types of devices the client should use the output parameter digits field to check that sufficient digits have been entered. The client should then get the user to re-enter their PIN with the correct number of digits.

If the client makes a call to [Pinpad.GetPinblock](#) or a local verification command without the minimum PIN digits having been entered, either the command will fail or the PIN verification will fail.

It is the responsibility of the client to identify the mapping between the FDK code and the physical location of the FDK.

Command Message

| Payload | Type | Required |
|-------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "minLen": 0, | integer | |
| "maxLen": 0, | integer | |
| "autoEnd": false, | boolean | |
| "echo": 0, | integer | |
| "activeFDKs": { | object | |
| "fdk01": false, | boolean | |
| "fdk02": false, | boolean | |
| "fdk03": false, | boolean | |
| "fdk04": false, | boolean | |

| | |
|-----------------|---------|
| "fdk05": false, | boolean |
| "fdk06": false, | boolean |
| "fdk07": false, | boolean |
| "fdk08": false, | boolean |
| "fdk09": false, | boolean |
| "fdk10": false, | boolean |
| "fdk11": false, | boolean |
| "fdk12": false, | boolean |
| "fdk13": false, | boolean |
| "fdk14": false, | boolean |
| "fdk15": false, | boolean |
| "fdk16": false, | boolean |
| "fdk17": false, | boolean |
| "fdk18": false, | boolean |
| "fdk19": false, | boolean |
| "fdk20": false, | boolean |
| "fdk21": false, | boolean |
| "fdk22": false, | boolean |
| "fdk23": false, | boolean |
| "fdk24": false, | boolean |
| "fdk25": false, | boolean |
| "fdk26": false, | boolean |
| "fdk27": false, | boolean |
| "fdk28": false, | boolean |
| "fdk29": false, | boolean |
| "fdk30": false, | boolean |

```
"fdk31": false,           boolean
"fdk32": false,           boolean
},
"activeKeys": {           object
  "fk0": false,           boolean
  "fk1": false,           boolean
  "fk2": false,           boolean
  "fk3": false,           boolean
  "fk4": false,           boolean
  "fk5": false,           boolean
  "fk6": false,           boolean
  "fk7": false,           boolean
  "fk8": false,           boolean
  "fk9": false,           boolean
  "fkA": false,           boolean
  "fkB": false,           boolean
  "fkC": false,           boolean
  "fkD": false,           boolean
  "fkE": false,           boolean
  "fkF": false,           boolean
  "fkEnter": false,        boolean
  "fkCancel": false,        boolean
  "fkClear": false,         boolean
  "fkBackspace": false,      boolean
  "fkHelp": false,          boolean
  "fkDecPoint": false,       boolean
```

```

"fk00": false,           boolean
"fk000": false,          boolean
"fkShift": false,         boolean
"fkRES01": false,        boolean
"fkRES02": false,        boolean
"fkRES03": false,        boolean
"fkRES04": false,        boolean
"fkRES05": false,        boolean
"fkRES06": false,        boolean
"fkRES07": false,        boolean
"fkRES08": false,        boolean
"fkOEM01": false,         boolean
"fkOEM02": false,         boolean
"fkOEM03": false,         boolean
"fkOEM04": false,         boolean
"fkOEM05": false,         boolean
"fkOEM06": false,         boolean
},
"terminateFDKs": {         object

```

See [activeFDKs](#) properties.

```

},
"terminateKeys": {         object

```

See [activeKeys](#) properties.

```

}
}
```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

minLen

Specifies the minimum number of digits which must be entered for the PIN. A value of zero indicates no minimum PIN length verification.

maxLen

Specifies the maximum number of digits which can be entered for the PIN. A value of zero indicates no maximum PIN length verification.

autoEnd

If autoEnd is set to true, the Service Provider terminates the command when the maximum number of digits are entered. Otherwise, the input is terminated by the user using one of the termination keys. autoEnd is ignored when maxLen is set to zero.

echo

Specifies the replace character to be echoed on a local display for the PIN digit.

activeFDKs

Specifies a mask of those FDKs which are active during the execution of the command

activeFDKs/fdk01

default: false

activeFDKs/fdk02

default: false

activeFDKs/fdk03

default: false

activeFDKs/fdk04

default: false

activeFDKs/fdk05

default: false

activeFDKs/fdk06

default: false

activeFDKs/fdk07

default: false

activeFDKs/fdk08

default: false

activeFDKs/fdk09

default: false

activeFDKs/fdk10

default: false

activeFDKs/fdk11

default: false

activeFDKs/fdk12

default: false

activeFDKs/fdk13

default: false

activeFDKs/fdk14

default: false

activeFDKs/fdk15

default: false

activeFDKs/fdk16

default: false

activeFDKs/fdk17

default: false

activeFDKs/fdk18

default: false

activeFDKs/fdk19

default: false

activeFDKs/fdk20

default: false

activeFDKs/fdk21

default: false

activeFDKs/fdk22

default: false

activeFDKs/fdk23

default: false

activeFDKs/fdk24

default: false

activeFDKs/fdk25

default: false

activeFDKs/fdk26

default: false

activeFDKs/fdk27

default: false

activeFDKs/fdk28

default: false

activeFDKs/fdk29

default: false

activeFDKs/fdk30

default: false

activeFDKs/fdk31

default: false

activeFDKs/fdk32

default: false

activeKeys

Specifies a mask of those (other) Function Keys which are active during the execution of the command

activeKeys/fk0

default: false

activeKeys/fk1

default: false

activeKeys/fk2

default: false

activeKeys/fk3

default: false

activeKeys/fk4

default: false

activeKeys/fk5

default: false

activeKeys/fk6

default: false

activeKeys/fk7

default: false

activeKeys/fk8

default: false

activeKeys/fk9

default: false

activeKeys/fkA

default: false

activeKeys/fkB

default: false

activeKeys/fkC

default: false

activeKeys/fkD

default: false

activeKeys/fkE

default: false

activeKeys/fkF

default: false

activeKeys/fkEnter

default: false

activeKeys/fkCancel

default: false

activeKeys/fkClear

default: false

activeKeys/fkBackspace

default: false

activeKeys/fkHelp

default: false

activeKeys/fkDecPoint

default: false

activeKeys/fk00

default: false

activeKeys/fk000

default: false

activeKeys/fkShift

default: false

activeKeys/fkRES01

default: false

activeKeys/fkRES02

default: false

activeKeys/fkRES03

default: false

activeKeys/fkRES04

default: false

activeKeys/fkRES05

default: false

activeKeys/fkRES06

default: false

activeKeys/fkRES07

default: false

activeKeys/fkRES08

default: false

activeKeys/fkOEM01

default: false

activeKeys/fkOEM02

default: false

activeKeys/fkOEM03

default: false

activeKeys/fkOEM04

default: false

activeKeys/fkOEM05

default: false

activeKeys/fkOEM06

default: false

terminateFDKs

Specifies a mask of those FDKs which must terminate the execution of the command

terminateKeys

Specifies a mask of those (other) Function Keys which must terminate the execution of the command

Completion Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "keyInvalid", | string | |
| "digits": 0, | integer | |
| "completion": "auto" | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- keyInvalid - At least one of the specified function keys or FDKs is invalid.
- keyNotSupported - At least one of the specified function keys or FDKs is not supported by the Service Provider.
- noActivekeys - There are no active function keys specified, or there is no defined layout definition.
- noTerminatekeys - There are no terminate keys specified and maxLen is not set to zero and autoEnd is false.
- minimumLength - The minimum PIN length field is invalid or greater than the maximum PIN length field when the maximum PIN length is not zero.
- tooManyFrames - The device requires that only one frame is used for this command.
- partialFrame - The single Touch Frame does not cover the entire monitor.
- entryTimeout - The timeout for entering data has been reached. This is a timeout which may be due to hardware limitations or legislative requirements (for example PCI).

digits

Specifies the number of PIN digits entered

completion

Specifies the reason for completion of the entry. Unless otherwise specified the following values must not be used in the execute event [Keyboard.PinEntry](#) or in the array of keys in the completion of [Keyboard.DataEntry](#)

Event Messages

- [Keyboard.KeyEvent](#)
- [Keyboard.EnterDataEvent](#)
- [Keyboard.LayoutEvent](#)

9.3.4 - Keyboard.DataEntry

This function enables keyboard insecure mode and report entered key in clear text with solicited events. For PIN pad device, this command will clear the pin unless the client has requested that the pin be maintained through the [Pinpad.MaintainPin](#) command.

Command Message

| Payload | Type | Required |
|-------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "maxLen": 0, | integer | |
| "autoEnd": false, | boolean | |
| "activeFDKs": { | object | |
| "fdk01": false, | boolean | |
| "fdk02": false, | boolean | |
| "fdk03": false, | boolean | |
| "fdk04": false, | boolean | |
| "fdk05": false, | boolean | |
| "fdk06": false, | boolean | |
| "fdk07": false, | boolean | |
| "fdk08": false, | boolean | |
| "fdk09": false, | boolean | |
| "fdk10": false, | boolean | |
| "fdk11": false, | boolean | |
| "fdk12": false, | boolean | |
| "fdk13": false, | boolean | |
| "fdk14": false, | boolean | |
| "fdk15": false, | boolean | |
| "fdk16": false, | boolean | |

```
"fdk17": false,           boolean
"fdk18": false,           boolean
"fdk19": false,           boolean
"fdk20": false,           boolean
"fdk21": false,           boolean
"fdk22": false,           boolean
"fdk23": false,           boolean
"fdk24": false,           boolean
"fdk25": false,           boolean
"fdk26": false,           boolean
"fdk27": false,           boolean
"fdk28": false,           boolean
"fdk29": false,           boolean
"fdk30": false,           boolean
"fdk31": false,           boolean
"fdk32": false           boolean
},
"activeKeys": {
"fk0": false,             boolean
"fk1": false,             boolean
"fk2": false,             boolean
"fk3": false,             boolean
"fk4": false,             boolean
"fk5": false,             boolean
"fk6": false,             boolean
"fk7": false              boolean
}
```

| | |
|-----------------------|---------|
| "fk8": false, | boolean |
| "fk9": false, | boolean |
| "fkA": false, | boolean |
| "fkB": false, | boolean |
| "fkC": false, | boolean |
| "fkD": false, | boolean |
| "fkE": false, | boolean |
| "fkF": false, | boolean |
| "fkEnter": false, | boolean |
| "fkCancel": false, | boolean |
| "fkClear": false, | boolean |
| "fkBackspace": false, | boolean |
| "fkHelp": false, | boolean |
| "fkDecPoint": false, | boolean |
| "fk00": false, | boolean |
| "fk000": false, | boolean |
| "fkShift": false, | boolean |
| "fkRES01": false, | boolean |
| "fkRES02": false, | boolean |
| "fkRES03": false, | boolean |
| "fkRES04": false, | boolean |
| "fkRES05": false, | boolean |
| "fkRES06": false, | boolean |
| "fkRES07": false, | boolean |
| "fkRES08": false, | boolean |
| "fkOEM01": false, | boolean |

```
"fkOEM02": false,          boolean  
"fkOEM03": false,          boolean  
"fkOEM04": false,          boolean  
"fkOEM05": false,          boolean  
"fkOEM06": false,          boolean  
},  
"terminateFDKs": {          object
```

See [activeFDKs](#) properties.

```
},  
"terminateKeys": {          object
```

See [activeKeys](#) properties.

```
}  
}
```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

maxLen

Specifies the maximum number of digits which can be returned to the client in the output parameter.

autoEnd

If autoEnd is set to true, the Service Provider terminates the command when the maximum number of digits are entered. Otherwise, the input is terminated by the user using one of the termination keys. autoEnd is ignored when maxLen is set to zero.

activeFDKs

Specifies a mask of those FDKs which are active during the execution of the command.

activeFDKs/fdk01

default: false

activeFDKs/fdk02

default: false

activeFDKs/fdk03

default: false

activeFDKs/fdk04

default: false

activeFDKs/fdk05

default: false

activeFDKs/fdk06

default: false

activeFDKs/fdk07

default: false

activeFDKs/fdk08

default: false

activeFDKs/fdk09

default: false

activeFDKs/fdk10

default: false

activeFDKs/fdk11

default: false

activeFDKs/fdk12

default: false

activeFDKs/fdk13

default: false

activeFDKs/fdk14

default: false

activeFDKs/fdk15

default: false

activeFDKs/fdk16

default: false

activeFDKs/fdk17

default: false

activeFDKs/fdk18

default: false

activeFDKs/fdk19

default: false

activeFDKs/fdk20

default: false

activeFDKs/fdk21

default: false

activeFDKs/fdk22

default: false

activeFDKs/fdk23

default: false

activeFDKs/fdk24

default: false

activeFDKs/fdk25

default: false

activeFDKs/fdk26

default: false

activeFDKs/fdk27

default: false

activeFDKs/fdk28

default: false

activeFDKs/fdk29

default: false

activeFDKs/fdk30

default: false

activeFDKs/fdk31

default: false

activeFDKs/fdk32

default: false

activeKeys

Specifies a mask of those (other) Function Keys which are active during the execution of the command.

activeKeys/fk0

default: false

activeKeys/fk1

default: false

activeKeys/fk2

default: false

activeKeys/fk3

default: false

activeKeys/fk4

default: false

activeKeys/fk5

default: false

activeKeys/fk6

default: false

activeKeys/fk7

default: false

activeKeys/fk8

default: false

activeKeys/fk9

default: false

activeKeys/fkA

default: false

activeKeys/fkB

default: false

activeKeys/fkC

default: false

activeKeys/fkD

default: false

activeKeys/fkE

default: false

activeKeys/fkF

default: false

activeKeys/fkEnter

default: false

activeKeys/fkCancel

default: false

activeKeys/fkClear

default: false

activeKeys/fkBackspace

default: false

activeKeys/fkHelp

default: false

activeKeys/fkDecPoint

default: false

activeKeys/fk00

default: false

activeKeys/fk000

default: false

activeKeys/fkShift

default: false

activeKeys/fkRES01

default: false

activeKeys/fkRES02

default: false

activeKeys/fkRES03

default: false

activeKeys/fkRES04

default: false

activeKeys/fkRES05

default: false

activeKeys/fkRES06

default: false

activeKeys/fkRES07

default: false

activeKeys/fkRES08

default: false

activeKeys/fkOEM01

default: false

activeKeys/fkOEM02

default: false

activeKeys/fkOEM03

default: false

activeKeys/fkOEM04

default: false

activeKeys/fkOEM05

default: false

activeKeys/fkOEM06

default: false

terminateFDKs

Specifies a mask of those FDKs which must terminate the execution of the command

terminateKeys

Specifies a mask of those (other) Function Keys which must terminate the execution of the command

Completion Message

| Payload | Type | Required |
|-----------------------------------|--------|----------|
| { "completionCode": "success", | string | |

```
"errorDescription": Add example to YAML, string  
"errorCode": "keyInvalid", string  
"keys": 0, integer  
"pinKeys": ["auto"], array (string)  
"completion": "auto" string  
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- keyInvalid - At least one of the specified function keys or FDKs is invalid.
- keyNotSupported - At least one of the specified function keys or FDKs is not supported by the Service Provider.
- noActivekeys - There are no active keys specified, or there is no defined layout definition.

keys

Number of keys entered by the user

pinKeys

Array to the pinKey that contain the keys entered by the user

completion

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Specifies the reason for completion of the entry

Event Messages

- [Keyboard.KeyEvent](#)
- [Keyboard.EnterDataEvent](#)
- [Keyboard.LayoutEvent](#)

[9.3.5 - Keyboard.Reset](#)

Sends a service reset to the Service Provider.

Command Message

| Payload | Type | Required |
|-------------------------------------|------|----------|
| { "timeout": 5000 integer } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|---|------|----------|
| { "completionCode": "success", string "errorDescription": Add example to YAML string } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

Event Messages

None

[9.3.6 - Keyboard.SecureKeyEntry](#)

This command allows a full length symmetric encryption key part to be entered directly into the device without being exposed outside of the device. From the point this function is invoked, encryption key digits (fk0 to fk9 and fkA to fkF) are not passed to the client. For each encryption key digit, or any other active key entered (except for shift), an execute notification event [Keyboard.KeyEvent](#) is sent in order to allow a client to perform the appropriate display action (i.e. when the device has no integrated display). When an encryption key digit is entered the client is not informed of the value entered, instead zero is returned.

The [Keyboard.EnterDataEvent](#) will be generated when the device is ready for the user to start entering data.

The keys that can be enabled by this command are defined by the FuncKeyDetail parameter of the [Keyboard.SecureKeyEntry](#) command. Function keys which are not associated with an encryption key digit may be enabled but will not contribute to the secure entry buffer (unless they are Cancel, Clear or Backspace) and will not count towards the length of the key entry. The Cancel and Clear keys will cause the encryption key buffer to be cleared. The Backspace key will cause the last encryption key digit in the encryption key buffer to be removed.

If autoEnd is TRUE the command will automatically complete when the required number of encryption key digits have been added to the buffer.

If autoEnd is FALSE then the command will not automatically complete and Enter, Cancel or any terminating key must be pressed. When keyLen hex encryption key digits have been

entered then all encryption key digits keys are disabled. If the Clear or Backspace key is pressed to reduce the number of entered encryption key digits below usKeyLen, the same keys will be reenabled.

Terminating keys have to be active keys to operate.

If an FDK is associated with Enter, Cancel, Clear or Backspace then the FDK must be activated to operate. The Enter and Cancel FDKs must also be marked as a terminator if they are to terminate entry. These FDKs are reported as normal FDKs within the KeyEvent, clients must be aware of those FDKs associated with Cancel, Clear, Backspace and Enter and handle any user interaction as required. For example, if the fdk01 is associated with Clear, then the client must include the fk_fdk01 FDK code in the activeFDKs parameter (if the clear functionality is required). In addition when this FDK is pressed the [Keyboard.KeyEvent](#) will contain the fk_fdk01 mask value in the digit field. The client must update the user interface to reflect the effect of the clear on the encryption key digits entered so far.

On some devices that are configured as either regularUnique or irregularUnique all the function keys on the device will be associated with hex digits and there may be no FDKs available either. On these devices there may be no way to correct mistakes or cancel the key encryption entry before all the encryption key digits are entered, so the client must set the autoEnd flag to TRUE and wait for the command to auto-complete. Clients should check the KCV to avoid storing an incorrect key component.

Encryption key parts entered with this command are stored through either the [KeyManagement.ImportKey](#). Each key part can only be stored once after which the secure key buffer will be cleared automatically.

Command Message

| Payload | Type | Required |
|-------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "keyLen": 0, | integer | |
| "autoEnd": false, | boolean | |
| "activeFDKs": { | object | |
| "fdk01": false, | boolean | |
| "fdk02": false, | boolean | |
| "fdk03": false, | boolean | |
| "fdk04": false, | boolean | |

| | |
|-----------------|---------|
| "fdk05": false, | boolean |
| "fdk06": false, | boolean |
| "fdk07": false, | boolean |
| "fdk08": false, | boolean |
| "fdk09": false, | boolean |
| "fdk10": false, | boolean |
| "fdk11": false, | boolean |
| "fdk12": false, | boolean |
| "fdk13": false, | boolean |
| "fdk14": false, | boolean |
| "fdk15": false, | boolean |
| "fdk16": false, | boolean |
| "fdk17": false, | boolean |
| "fdk18": false, | boolean |
| "fdk19": false, | boolean |
| "fdk20": false, | boolean |
| "fdk21": false, | boolean |
| "fdk22": false, | boolean |
| "fdk23": false, | boolean |
| "fdk24": false, | boolean |
| "fdk25": false, | boolean |
| "fdk26": false, | boolean |
| "fdk27": false, | boolean |
| "fdk28": false, | boolean |
| "fdk29": false, | boolean |
| "fdk30": false, | boolean |

```
"fdk31": false,           boolean
"fdk32": false,           boolean
},
"activeKeys": {           object
  "fk0": false,           boolean
  "fk1": false,           boolean
  "fk2": false,           boolean
  "fk3": false,           boolean
  "fk4": false,           boolean
  "fk5": false,           boolean
  "fk6": false,           boolean
  "fk7": false,           boolean
  "fk8": false,           boolean
  "fk9": false,           boolean
  "fkA": false,           boolean
  "fkB": false,           boolean
  "fkC": false,           boolean
  "fkD": false,           boolean
  "fkE": false,           boolean
  "fkF": false,           boolean
  "fkEnter": false,        boolean
  "fkCancel": false,        boolean
  "fkClear": false,         boolean
  "fkBackspace": false,      boolean
  "fkHelp": false,          boolean
  "fkDecPoint": false,       boolean
```

```

"fk00": false,           boolean
"fk000": false,          boolean
"fkShift": false,         boolean
"fkRES01": false,         boolean
"fkRES02": false,         boolean
"fkRES03": false,         boolean
"fkRES04": false,         boolean
"fkRES05": false,         boolean
"fkRES06": false,         boolean
"fkRES07": false,         boolean
"fkRES08": false,         boolean
"fkOEM01": false,          boolean
"fkOEM02": false,          boolean
"fkOEM03": false,          boolean
"fkOEM04": false,          boolean
"fkOEM05": false,          boolean
"fkOEM06": false,          boolean
},
"terminateFDKs": {         object

```

See [activeFDKs](#) properties.

```
},
"trerminateKeys": {        object

```

See [activeKeys](#) properties.

```
},
"verificationType": "self"   string
}

```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

keyLen

Specifies the number of digits which must be entered for the encryption key, 16 for a singlelength key, 32 for a double-length key and 48 for a triple-length key. The only valid values are 16, 32 and 48.

autoEnd

If autoEnd is set to true, the Service Provider terminates the command when the maximum number of encryption key digits are entered. Otherwise, the input is terminated by the user using Enter, Cancel or any terminating key. When keyLen is reached, the Service Provider will disable all keys associated with an encryption key digit.

default: false

activeFDKs

Specifies those FDKs which are active during the execution of the command. This parameter should include those FDKs mapped to edit functions.

activeFDKs/fdk01

default: false

activeFDKs/fdk02

default: false

activeFDKs/fdk03

default: false

activeFDKs/fdk04

default: false

activeFDKs/fdk05

default: false

activeFDKs/fdk06

default: false

activeFDKs/fdk07

default: false

activeFDKs/fdk08

default: false

activeFDKs/fdk09

default: false

activeFDKs/fdk10

default: false

activeFDKs/fdk11

default: false

activeFDKs/fdk12

default: false

activeFDKs/fdk13

default: false

activeFDKs/fdk14

default: false

activeFDKs/fdk15

default: false

activeFDKs/fdk16

default: false

activeFDKs/fdk17

default: false

activeFDKs/fdk18

default: false

activeFDKs/fdk19

default: false

activeFDKs/fdk20

default: false

activeFDKs/fdk21

default: false

activeFDKs/fdk22

default: false

activeFDKs/fdk23

default: false

activeFDKs/fdk24

default: false

activeFDKs/fdk25

default: false

activeFDKs/fdk26

default: false

activeFDKs/fdk27

default: false

activeFDKs/fdk28

default: false

activeFDKs/fdk29

default: false

activeFDKs/fdk30

default: false

activeFDKs/fdk31

default: false

activeFDKs/fdk32

default: false

activeKeys

Specifies all Function Keys(not FDKs) which are active during the execution of the command. This should be the complete set or a subset of the keys returned in the funcKeyDetail parameter of the SecureKeyDetail command.

activeKeys/fk0

default: false

activeKeys/fk1

default: false

activeKeys/fk2

default: false

activeKeys/fk3

default: false

activeKeys/fk4

default: false

activeKeys/fk5

default: false

activeKeys/fk6

default: false

activeKeys/fk7

default: false

activeKeys/fk8

default: false

activeKeys/fk9

default: false

activeKeys/fkA

default: false

activeKeys/fkB

default: false

activeKeys/fkC

default: false

activeKeys/fkD

default: false

activeKeys/fkE

default: false

activeKeys/fkF

default: false

activeKeys/fkEnter

default: false

activeKeys/fkCancel

default: false

activeKeys/fkClear

default: false

activeKeys/fkBackspace

default: false

activeKeys/fkHelp

default: false

activeKeys/fkDecPoint

default: false

activeKeys/fk00

default: false

activeKeys/fk000

default: false

activeKeys/fkShift

default: false

activeKeys/fkRES01

default: false

activeKeys/fkRES02

default: false

activeKeys/fkRES03

default: false

activeKeys/fkRES04

default: false

activeKeys/fkRES05

default: false

activeKeys/fkRES06

default: false

activeKeys/fkRES07

default: false

activeKeys/fkRES08

default: false

activeKeys/fkOEM01

default: false

activeKeys/fkOEM02

default: false

activeKeys/fkOEM03

default: false

activeKeys/fkOEM04

default: false

activeKeys/fkOEM05

default: false

activeKeys/fkOEM06

default: false

terminateFDKs

Specifies those FDKs which must terminate the execution of the command. This should include the FDKs associated with Cancel and Enter.

terminateKeys

Specifies those all Function Keys (not FDKs) which must terminate the execution of the command. This does not include the FDKs associated with Enter or Cancel.

verificationType

Specifies the type of verification to be done on the entered key.

Completion Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "accessDenied", | string | |
| "digits": 0, | integer | |
| "completion": "auto", | string | |
| "kcv": Add example to YAML | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- accessDenied - The encryption module is either not initialized or not ready for any vendor specific reason.
- keyInvalid - At least one of the specified function keys or FDKs is invalid.
- keyNotSupported - At least one of the specified function keys or FDKs is not supported by the Service Provider.
- noActiveKeys - There are no active function keys specified, or there is no defined layout definition.
- noTerminatekeys - There are no terminate keys specified and autoEnd is false.
- invalidKeyLength - The keyLen key length is not supported.
- modeNotSupported - The KCV mode is not supported.
- tooManyFrames - The device requires that only one frame is used for this command.
- partialFrame - The single Touch Frame does not cover the entire monitor.
- missingKeys - The single frame does not contain a full set of hexadecimal key definitions.
- entryTimeout - The timeout for entering data has been reached. This is a timeout which may be due to hardware limitations or legislative requirements (for example PCI).

digits

Specifies the number of key digits entered. Clients must ensure all required digits have been entered before trying to store the key.

completion

Specifies the reason for completion of the entry.

kcv

Contains the key check value data that can be used for verification of the entered key formatted in base 64. This field is not set if device does not have this capability, or the key entry was not fully entered, e.g. the entry was terminated by Enter before the required number of digits was entered.

Event Messages

- [Keyboard.KeyEvent](#)
- [Keyboard.EnterDataEvent](#)
- [Keyboard.LayoutEvent](#)

[9.3.7 - Keyboard.KeypressBeep](#)

This command is used to enable or disable the device from emitting a beep tone on subsequent key presses of active or in-active keys. This command is valid only on devices which have the capability to support client control of automatic beeping. See [Keyboard.Capabilities](#) structure for information.

Command Message

| Payload | Type | Required |
|------------------------------|--------|----------|
| { | | |
| "timeout": 5000, integer | | |
| "mode": { | object | |
| "active": false, boolean | | |
| "inactive": false boolean | | |
| } | | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

mode

Specifies whether automatic generation of key press beep tones should be activated for any active or in-active key subsequently pressed on the PIN. mode selectively turns beeping on and off for active, -active or both types of keys.

mode/active

Specifies that beeping should be enabled for active keys. If this flag is not present then beeping is disabled for active keys.

default: false

mode/inactive

Specifies that beeping should be enabled for in-active keys. If this flag is not present then beeping is disabled for in-active keys.

default: false

Completion Message

| Payload | Type | Required |
|---|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML | string | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional

information

Event Messages

None

9.3.8 - Keyboard.DefineLayout

This command allows a client to configure a layout for any device. One or more layouts can be defined with a single request of this command.

There can be a layout for each of the different types of keyboard entry modes, if the vendor and the hardware supports these different methods. The types of keyboard entry modes are (1) Mouse mode,(2) Data mode which corresponds to the [Keyboard.DataEntry](#) command, (3) PIN mode which corresponds to the [Keyboard.PinEntry](#) command,(4) Secure mode which corresponds to the [Keyboard.SecureKeyEntry](#) command. One or more layouts can be preloaded into the device, if the device supports this, or a single layout can be loaded into the device immediately prior to the keyboard command being requested.

If a [Keyboard.DataEntry](#), [Keyboard.PinEntry](#), or [Keyboard.SecureKeyEntry](#) command is already in progress (or queued), then this command is rejected with a command result of SequenceError.

Layouts defined with this command are persistent.

Command Message

| Payload | Type | Required |
|-----------------|----------------|----------|
| { | | |
| "entryMode": { | object | |
| "data": false, | boolean | |
| "pin": false, | boolean | |
| "secure": false | boolean | |
| }, | | |
| "frames": [{ | array (object) | |
| "xPos": 0, | integer | |
| "yPos": 0, | integer | |

```

"xSize": 0,           integer
"ySize": 0,           integer
"floatAction": {     object
  "floatX": false,   boolean
  "floatY": false,   boolean
},
"flks": [{            array (object)
  "xPos": 0,         integer
  "yPos": 0,         integer
  "xSize": 0,         integer
  "ySize": 0,         integer
  "fk": "fk0",       string
  "shiftFK": "fk0"  string
}]
}
}
}

```

Properties

entryMode

Specifies entry mode to be returned. It can be one of the following flags, or zero to return all supported entry modes.

entryMode/data

Specifies that the layout be applied to the [Keyboard.DataEntry](#) method.

default: false

entryMode/pin

Specifies that the layout be applied to the [Keyboard.PinEntry](#) method.

default: false

entryMode/secure

Specifies that the layout be applied to the [Keyboard.SecurekeyEntry](#) method.

default: false

frames

There can be one or more frame structures included

frames/xPos

For ETS, specifies the left coordinate of the frame as an offset from the left edge of the screen. For all other device types, this value is ignored

frames/yPos

For ETS, specifies the top coordinate of the frame as an offset from the top edge of the screen. For all other device types, this value is ignored

frames/xSize

For ETS, specifies the width of the frame. For all other device types, this value is ignored

frames/ySize

For ETS, specifies the height of the frame. For all other device types, this value is ignored

frames/floatAction

Specifies if the device can float the touch keyboards

frames/floatAction/floatX

Specifies that the PIN device will randomly shift the layout in a horizontal direction

default: false

frames/floatAction/floatY

Specifies that the PIN device will randomly shift the layout in a vertical direction

default: false

frames/fks

Defining details of the keys in the keyboard.

frames/fks/xPos

Specifies the position of the top left corner of the FK relative to the left hand side of the layout. For ETS devices, must be in the range defined in the frame. For non-ETS devices, must be a value between 0 and 999, where 0 is the left edge and 999 is the right edge.

frames/fks/yPos

Specifies the position of the top left corner of the FK relative to the left hand side of the layout. For ETS devices, must be in the range defined in the frame. For non-ETS devices, must be a value between 0 and 999, where 0 is the top edge and 999 is the bottom edge.

frames/fks/xSize

Specifies the FK width. For ETS, width is measured in pixels. For non-ETS devices, width is expressed as a value between 1 and 1000, where 1 is the smallest possible size and 1000 is the full width of the layout.

frames/fks/ySize

Specifies the FK height. For ETS, height is measured in pixels. For non-ETS devices, height is expressed as a value between 1 and 1000, where 1 is the smallest possible size and 1000 is the full height of the layout.

frames/fks/fk

Specifies the FK code associated with the physical area in non-shifted mode.

frames/fks/shiftFK

Specifies the FK code associated with the physical key in shifted mode.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "modeNotSupported" | string | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- modeNotSupported - The device does not support the float action.
- frameCoordinate - A frame coordinate or size field is out of range.
- keyCoordinate - A key coordinate or size field is out of range.
- frameOverlap - Frames are overlapping.
- keyOverlap - Keys are overlapping.
- tooManyFrames -There are more frames defined than allowed.
- tooManyKeys - There are more keys defined than allowed.
- keyAlreadyDefined - The combination of the wKeyType and values for fK and shiftFK can only be used once per layout.

Event Messages

None

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

9.4 - Event Messages

9.4.1 - Keyboard.KeyEvent

This event specifies that any active key has been pressed at the PIN pad. It is used if the device has no internal display unit and the client has to manage the display of the entered digits. It is the responsibility of the client to identify the mapping between the FDK code and the physical location of the FDK.

| Payload | Type | Required |
|-----------------------|---------|----------|
| { | | |
| "completion": "auto", | string | |
| "digit": { | object | |
| "fk0": false, | boolean | |
| "fk1": false, | boolean | |
| "fk2": false, | boolean | |
| "fk3": false, | boolean | |
| "fk4": false, | boolean | |
| "fk5": false, | boolean | |
| "fk6": false, | boolean | |
| "fk7": false, | boolean | |
| "fk8": false, | boolean | |
| "fk9": false, | boolean | |
| "fkA": false, | boolean | |
| "fkB": false, | boolean | |
| "fkC": false, | boolean | |
| "fkD": false, | boolean | |
| "fkE": false, | boolean | |
| "fkF": false, | boolean | |

| | |
|-----------------------|---------|
| "fkEnter": false, | boolean |
| "fkCancel": false, | boolean |
| "fkClear": false, | boolean |
| "fkBackspace": false, | boolean |
| "fkHelp": false, | boolean |
| "fkDecPoint": false, | boolean |
| "fk00": false, | boolean |
| "fk000": false, | boolean |
| "fkShift": false, | boolean |
| "fkRES01": false, | boolean |
| "fkRES02": false, | boolean |
| "fkRES03": false, | boolean |
| "fkRES04": false, | boolean |
| "fkRES05": false, | boolean |
| "fkRES06": false, | boolean |
| "fkRES07": false, | boolean |
| "fkRES08": false, | boolean |
| "fkOEM01": false, | boolean |
| "fkOEM02": false, | boolean |
| "fkOEM03": false, | boolean |
| "fkOEM04": false, | boolean |
| "fkOEM05": false, | boolean |
| "fkOEM06": false, | boolean |
| "fdk01": false, | boolean |
| "fdk02": false, | boolean |
| "fdk03": false, | boolean |

| | |
|-----------------|---------|
| "fdk04": false, | boolean |
| "fdk05": false, | boolean |
| "fdk06": false, | boolean |
| "fdk07": false, | boolean |
| "fdk08": false, | boolean |
| "fdk09": false, | boolean |
| "fdk10": false, | boolean |
| "fdk11": false, | boolean |
| "fdk12": false, | boolean |
| "fdk13": false, | boolean |
| "fdk14": false, | boolean |
| "fdk15": false, | boolean |
| "fdk16": false, | boolean |
| "fdk17": false, | boolean |
| "fdk18": false, | boolean |
| "fdk19": false, | boolean |
| "fdk20": false, | boolean |
| "fdk21": false, | boolean |
| "fdk22": false, | boolean |
| "fdk23": false, | boolean |
| "fdk24": false, | boolean |
| "fdk25": false, | boolean |
| "fdk26": false, | boolean |
| "fdk27": false, | boolean |
| "fdk28": false, | boolean |
| "fdk29": false, | boolean |

```
"fdk30": false,      boolean  
"fdk31": false,      boolean  
"fdk32": false,      boolean  
}  
}
```

Properties

completion

Specifies the reason for completion of the entry.

digit

Specifies the digit entered by the user. When working in encryption mode or secure key entry mode ([Keyboard.PinEntry](#) and [Keyboard.SecureKeyEntry](#)), the value of this field is 0x00 for the function keys 0-9 and A-F. Otherwise, for each key pressed, the corresponding FK or FDK mask value is stored in this field.

digit/fk0

default: false

digit/fk1

default: false

digit/fk2

default: false

digit/fk3

default: false

digit/fk4

default: false

digit/fk5

default: false

digit/fk6

default: false

digit/fk7

default: false

digit/fk8

default: false

digit/fk9

default: false

digit/fkA

default: false

digit/fkB

default: false

digit/fkC

default: false

digit/fkD

default: false

digit/fkE

default: false

digit/fkF

default: false

digit/fkEnter

default: false

digit/fkCancel

default: false

digit/fkClear

default: false

digit/fkBackspace

default: false

digit/fkHelp

default: false

digit/fkDecPoint

default: false

digit/fk00

default: false

digit/fk000

default: false

digit/fkShift

default: false

digit/fkRES01

default: false

digit/fkRES02

default: false

digit/fkRES03

default: false

digit/fkRES04

default: false

digit/fkRES05

default: false

digit/fkRES06

default: false

digit/fkRES07

default: false

digit/fkRES08

default: false

digit/fkOEM01

default: false

digit/fkOEM02

default: false

digit/fkOEM03

default: false

digit/fkOEM04

default: false

digit/fkOEM05

default: false

digit/fkOEM06

default: false

digit/fdk01

default: false

digit/fdk02

default: false

digit/fdk03

default: false

digit/fdk04

default: false

digit/fdk05

default: false

digit/fdk06

default: false

digit/fdk07

default: false

digit/fdk08

default: false

digit/fdk09

default: false

digit/fdk10

default: false

digit/fdk11

default: false

digit/fdk12

default: false

digit/fdk13

default: false

digit/fdk14

default: false

digit/fdk15

default: false

digit/fdk16

default: false

digit/fdk17

default: false

digit/fdk18

default: false

digit/fdk19

default: false

digit/fdk20

default: false

digit/fdk21

default: false

digit/fdk22

default: false

digit/fdk23

default: false

digit/fdk24

default: false

digit/fdk25

default: false

digit/fdk26

default: false

digit/fdk27

default: false

digit/fdk28

default: false

digit/fdk29

default: false

digit/fdk30

default: false

digit/fdk31

default: false

digit/fdk32

default: false

[9.4.2 - Keyboard.EnterDataEvent](#)

This mandatory event notifies the client when the device is ready for the user to start entering data.

9.4.3 - Keyboard.LayoutEvent

This event sends the layout for a specific keyboard entry mode if the layout has changed since it was loaded (i.e. if a float action is being used).

| Payload | Type | Required |
|------------------|----------------|----------|
| { | | |
| "entryMode": { | object | |
| "data": false, | boolean | |
| "pin": false, | boolean | |
| "secure": false | boolean | |
| }, | | |
| "frames": [{ | array (object) | |
| "xPos": 0, | integer | |
| "yPos": 0, | integer | |
| "xSize": 0, | integer | |
| "ySize": 0, | integer | |
| "floatAction": { | object | |
| "floatX": false, | boolean | |
| "floatY": false | boolean | |
| }, | | |
| "fks": [{ | array (object) | |
| "xPos": 0, | integer | |
| "yPos": 0, | integer | |
| "xSize": 0, | integer | |
| "ySize": 0, | integer | |
| "fk": "fk0", | string | |
| "shiftFK": "fk0" | string | |
| }] | | |
| }] | | |

}

Properties

entryMode

Specifies entry mode to be returned. It can be one of the following flags, or zero to return all supported entry modes.

entryMode/data

Specifies that the layout be applied to the [Keyboard.DataEntry](#) method.

default: false

entryMode/pin

Specifies that the layout be applied to the [Keyboard.PinEntry](#) method.

default: false

entryMode/secure

Specifies that the layout be applied to the [Keyboard.SecurekeyEntry](#) method.

default: false

frames

There can be one or more frame structures included

frames/xPos

For ETS, specifies the left coordinate of the frame as an offset from the left edge of the screen. For all other device types, this value is ignored

frames/yPos

For ETS, specifies the top coordinate of the frame as an offset from the top edge of the screen. For all other device types, this value is ignored

frames/xSize

For ETS, specifies the width of the frame. For all other device types, this value is ignored

frames/ySize

For ETS, specifies the height of the frame. For all other device types, this value is ignored

frames/floatAction

Specifies if the device can float the touch keyboards

frames/floatAction/floatX

Specifies that the PIN device will randomly shift the layout in a horizontal direction

default: false

frames/floatAction/floatY

Specifies that the PIN device will randomly shift the layout in a vertical direction

default: false

frames/fks

Defining details of the keys in the keyboard.

frames/fks/xPos

Specifies the position of the top left corner of the FK relative to the left hand side of the layout. For ETS devices, must be in the range defined in the frame. For non-ETS devices, must be a value between 0 and 999, where 0 is the left edge and 999 is the right edge.

frames/fks/yPos

Specifies the position of the top left corner of the FK relative to the left hand side of the layout. For ETS devices, must be in the range defined in the frame. For non-ETS devices, must be a value between 0 and 999, where 0 is the top edge and 999 is the bottom edge.

frames/fks/xSize

Specifies the FK width. For ETS, width is measured in pixels. For non-ETS devices, width is expressed as a value between 1 and 1000, where 1 is the smallest possible size and 1000 is the full width of the layout.

frames/fks/ySize

Specifies the FK height. For ETS, height is measured in pixels. For non-ETS devices, height is expressed as a value between 1 and 1000, where 1 is the smallest possible size and 1000 is the full height of the layout.

frames/fks/flk

Specifies the FK code associated with the physical area in non-shifted mode.

frames/fks/shiftFK

Specifies the FK code associated with the physical key in shifted mode.

10 - Pinpad Interface

This chapter defines the Pinpad interface functionality and messages.

10.1 - Summary

This section describes the general interface for the following functions:

- Administration of encryption devices
- PIN verification
- PIN block generation (encrypted PIN)
- PIN presentation to chipcard
- EMV 4.0 PIN blocks, EMV 4.0 public key loading, static and dynamic data verification

10.2 - General Information

10.2.1 - DUKPT

Definitions and Abbreviations

| | |
|-------|------------------------------------|
| DUKPT | Derived Unique Key Per Transaction |
| BDK | Base Derivation Key |
| IPEK | Initial PIN Encryption Key |
| KSN | Key Serial Number. |
| TRSM | Tamper Resistant Security Module. |

2.1 Default Key Name

The dukpt IPEK key is given a fixed name so multi-vendor clients can be developed without the need for vendor specific configuration tools.

If dukpt is supported, this key must be included in the KeyDetail output.

| Item Name | Description |
|-------------|--|
| “dukptIpek” | This key represents the IPEK, the derived future keys stored during import of the IPEK and the variant per transaction keys (PIN and optionally data and MAC). |

10.3 - Command Messages

10.3.1 - Pinpad.GetQueryPCIPTSDeviceId

This command is used to report information in order to verify the PCI Security Standards Council PIN transaction security (PTS) certification held by the PIN device. The command provides detailed information in order to verify the certification level of the device. Support of this command by the Service Provider does not imply in anyway the certification level achieved by the device.

Command Message

| Payload | Type | Required |
|--------------------------------------|------|----------|
| { "timeout": 5000 integer } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "manufacturerIdentifier": Add example to YAML, | string | |
| "modelIdentifier": Add example to YAML, | string | |
| "hardwareIdentifier": Add example to YAML, | string | |

```
"firmwareIdentifier": Add example to YAML,      string  
"clientIdentifier": Add example to YAML        string  
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

manufacturerIdentifier

Returns an ASCII string containing the manufacturer identifier of the PIN device. This value is not set if the manufacturer identifier is not available. This field is distinct from the hsm key pair that may be reported in the extra field by the [Crypto.Capabilities](#) command.

modelIdentifier

Returns an ASCII string containing the model identifier of the PIN device. This value is not set if the model identifier is not available.

hardwareIdentifier

Returns an ASCII string containing the hardware identifier of the PIN device. This value is not set if the hardware identifier is not available.

firmwareIdentifier

Returns an ASCII string containing the firmware identifier of the PIN device. This value is not set if the firmware identifier is not available.

clientIdentifier

Returns an ASCII string containing the client identifier of the PIN device. This value is not set if the client identifier is not available.

Event Messages

None

[10.3.2 - Pinpad.LocalDES](#)

The PIN, which was entered with the GetPin command, is combined with the requisite data specified by the DES validation algorithm and locally verified for correctness. The result of the verification is returned to the client. This command will clear the PIN unless the client has requested that the PIN be maintained through the [Pinpad.MaintainPin](#) command.

Command Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "validationData": Add example to YAML, | string | |
| "offset": Add example to YAML, | string | |
| "padding": Add example to YAML, | string | |
| "maxPIN": 0, | integer | |
| "valDigits": 0, | integer | |
| "noLeadingZero": false, | boolean | |
| "key": Add example to YAML, | string | |
| "keyEncKey": Add example to YAML, | string | |
| "decTable": Add example to YAML | string | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

validationData

Customer specific data (normally obtained from card track data) used to validate the correctness of the PIN. The validation data should be an ASCII string.

offset

ASCII string defining the offset data for the PIN block as an ASCII string. If this field is not set then no offset is used. The character must be in the ranges '0' to '9', 'a' to 'f' and 'A' to 'F'.
pattern: "^[0-9a-fA-F]\$"

padding

Specifies the padding character for the validation data. If the validation data is less than 16 characters long then it will be padded with this character. If padding is in the range 00 to 0F in 16 character string, padding is applied after the validation data has been compressed. If the padding character is in the range 30 to 39 ('0' to '9' in ASCII code), 41 to 46 ('A' to 'F' in ASCII code), or 61 to 66 ('a' to 'f' in ASCII code) in hexadecimal string format, padding is applied before the validation data is compressed.
pattern: "^0[0-9a-fA-F]\$|^3[0-9]\$|^4[1-6]\$|^6[1-6]\$"

maxPIN

Maximum number of PIN digits to be used for validation. This parameter corresponds to PINMINL in the IBM 3624 specification.

valDigits

Number of Validation digits from the validation data to be used for validation. This is the length of the validationData string.

noLeadingZero

If set to TRUE and the first digit of result of the modulo 10 addition is a 0x0, it is replaced with 0x1 before performing the verification against the entered PIN. If set to FALSE, a

leading zero is allowed in entered PINs.

key

Name of the key to be used for validation. The key referenced by key must have the function or pinLocal attribute.

keyEncKey

If this field is not set, key is used directly for PIN validation. Otherwise, key is used to decrypt the encrypted key passed in keyEncKey and the result is used for PIN validation.

decTable

ASCII decimalization table (16 character string containing characters '0' to '9'). This table is used to convert the hexadecimal digits (0x0 to 0xF) of the encrypted validation data to decimal digits (0x0 to 0x9).

Completion Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "keyNotFound", | string | |
| "result": false | boolean | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- keyNotFound - The specified key was not found.
- accessDenied - The encryption module is either not initialized or not ready for any vendor specific reason.
- keyNoValue - The specified key name was found but the corresponding key value has not been loaded.
- useViolation - The use specified by keyUsage is not supported.
- noPin - The PIN has not been entered was not long enough or has been cleared.
- formatNotSupported - The specified format is not supported.
- invalidKeyLength - The length of keyEncKey is not supported or the length of an encryption key is not compatible with the encryption operation required.

result

boolean value which specifies whether the PIN is correct or not.

Event Messages

None

[10.3.3 - Pinpad.LocalVisa](#)

The PIN, which was entered with the GetPin command, is combined with the requisite data specified by the VISA validation algorithm and locally verified for correctness. The result of the verification is returned to the client.

This command will clear the PIN unless the client has requested that the PIN be maintained through the [Pinpad.MaintainPin](#) command.

Command Message

| Payload | Type | Required |
|------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |

```
"pan": Add example to YAML,      string
"pvv": Add example to YAML,      string
"pvvDigits": 0,                  integer
"key": Add example to YAML,      string
"keyEncKey": Add example to YAML string
}
}
```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

pan

Primary Account Number from track data, as an ASCII string. PAN should contain the eleven rightmost digits of the PAN (excluding the check digit), followed by the pvki indicator in the 12th byte.

pvv

PIN Validation Value from track data, as an ASCII string with characters in the range '0' to '9'. This string should contain 4 digits.

pvvDigits

Number of digits of PVV.

key

Name of the validation key. The key referenced by key must have the function or pinLocal attribute

keyEncKey

If this field is not set, key is used directly for PIN validation. Otherwise, key is used to decrypt the encrypted key passed in keyEncKey and the result is used for PIN validation.

Completion Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "keyNotFound", | string | |
| "result": false | boolean | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- keyNotFound - The specified key was not found.
- accessDenied - The encryption module is either not initialized or not ready for any vendor specific reason.
- keyNoValue - The specified key name was found but the corresponding key value has not been loaded.
- useViolation - The use specified by keyUsage is not supported.
- noPin - The PIN has not been entered was not long enough or has been cleared.

- `formatNotSupported` - The specified format is not supported.
- `invalidKeyLength` - The length of `keyEncKey` is not supported or the length of an encryption key is not compatible with the encryption operation required.

result

Pointer to a boolean value which specifies whether the PIN is correct or not.

Event Messages

None

10.3.4 - Pinpad.PresentIDC

The PIN, which was entered with the `GetPin` command, is combined with the requisite data specified by the IDC presentation algorithm and presented to the smartcard contained in the ID card unit. The result of the presentation is returned to the client. This command will clear the PIN unless the client has requested that the PIN be maintained through the `Pinpad.MaintainPin` command.

Command Message

| Payload | Type | Required |
|---|---------|----------|
| { | | |
| <code>"timeout": 5000,</code> | integer | |
| <code>"presentAlgorithm": "presentClear",</code> | string | |
| <code>"chipProtocol": Add example to YAML,</code> | string | |
| <code>"chipData": Add example to YAML,</code> | string | |
| <code>"algorithmData": {</code> | object | |
| <code>"pinPointer": 0,</code> | integer | |
| <code>"pinOffset": 0</code> | integer | |
| <code>}</code> | | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

presentAlgorithm

Specifies the algorithm that is used for presentation. Possible values are: (see command [Pinpad.Capabilities](#)).

chipProtocol

Identifies the protocol that is used to communicate with the chip. Possible values are: (see command [CardReader.Capabilities](#) in the Identification Card Device Class Interface)

chipData

Points to the data to be sent to the chip formatted in base64."

algorithmData

Contains the data required for the specified presentation algorithm

algorithmData/pinPointer

The byte offset where to start inserting the PIN into chipData. The leftmost byte is numbered zero. See below for an example

algorithmData/pinOffset

The bit offset within the byte specified by pinPointer where to start inserting the PIN. The leftmost bit numbered zero.

Completion Message

| Payload | Type | Required |
|---------|------|----------|
|---------|------|----------|

```
{  
  "completionCode": "success",           string  
  "errorDescription": Add example to YAML, string  
  "errorCode": "accessDenied",          string  
  "chipProtocol": Add example to YAML,    string  
  "chipData": Add example to YAML        string  
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- accessDenied - The encryption module is either not initialized or not ready for any vendor specific reason.
- noPin - The PIN has not been entered was not long enough or has been cleared.
- protocolNotSupported - The specified protocol is not supported by the Service Provider.
- invalidData - An error occurred while communicating with the chip.

chipProtocol

Identifies the protocol that was used to communicate with the chip. This field contains the same value as the corresponding field in the input.

chipData

The data responded from the chip.

Event Messages

None

10.3.5 - Pinpad.Reset

Sends a service reset to the Service Provider.

Command Message

| Payload | Type | Required |
|---------------------------|---------|----------|
| { "timeout": 5000 } | integer | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|---|--------|----------|
| { "completionCode": "success", "errorDescription": Add example to YAML } | string | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

Event Messages

None

[10.3.6 - Pinpad.MaintainPin](#)

This command is used to control if the PIN is maintained after a PIN processing command for subsequent use by other PIN processing commands. This command is also used to clear the PIN buffer when the PIN is no longer required.

Command Message

| Payload | Type | Required |
|----------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "maintainPIN": false | boolean | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

maintainPIN

Specifies if the PIN should be maintained after a PIN processing command. Once set, this setting applies until changed through another call to this command

default: false

Completion Message

| <u>Payload</u> | <u>Type</u> | <u>Required</u> |
|---|-------------|-----------------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML | string | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

Event Messages

None

10.3.7 - Pinpad.SetPinBlockData

This function should be used for devices which need to know the data for the PIN block before the PIN is entered by the user. [Keyboard.GetPin](#) and [Pinpad.GetPinBlock](#) should be called after this command. For all other devices Unsupported will be returned here. If this command is required and it is not called, the [Keyboard.GetPin](#) command will fail with the

generic error SequenceError. If the input parameters passed to this command and [Pinpad.GetPinBlock](#) are not identical, the [Pinpad.GetPinBlock](#) command will fail with the generic error InvalidData. The data associated with this command will be cleared on a [Pinpad.GetPinBlock](#) command.

Command Message

| Payload | Type | Required |
|--------------------------------------|---------|----------|
| { | | |
| "errorCode": "keyNotFound", | string | |
| "timeout": 5000, | integer | |
| "customerData": Add example to YAML, | string | |
| "xorData": Add example to YAML, | string | |
| "padding": 0, | integer | |
| "format": "ibm3624", | string | |
| "key": Add example to YAML, | string | |
| "secondEncKey": Add example to YAML, | string | |
| "pinBlockAttributes": { | object | |
| "algorithm": "A", | string | |
| "cryptoMethod": "ecb" | string | |
| } | | |
| } | | |

Properties

errorCode

Specifies the error code if applicable. The following values are possible:

- keyNotFound - The specified key was not found.
- accessDenied - The encryption module is either not initialized or not ready for any vendor specific reason.
- keyNoValue - The specified key name was found but the corresponding key value has not been loaded.

- useViolation - The use specified by keyUsage is not supported.
- noPin - The PIN has not been entered was not long enough or has been cleared.
- formatNotSupported - The specified format is not supported.
- invalidKeyLength - The length of keyEncKey or key is not supported by this key or the length of an encryption key is not compatible with the encryption operation required.

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

customerData

The customer data should be an ASCII string. Used for ANSI, ISO-0 and ISO-1 algorithm to build the formatted PIN. For ANSI and ISO-0 the PAN (Primary Account Number, without the check number) is supplied, for ISO-1 a ten digit transaction field is required. If not used a NULL is required. Used for DIBOLD with coordination number, as a two digit coordination number. Used for EMV with challenge number (8 bytes) coming from the chip card. This number is passed as unpacked string, for example: 0123456789ABCDEF = 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x41 0x42 0x43 0x44 0x45 0x46 For AP PIN blocks, the data must be a concatenation of the PAN (18 digits including the check digit), and the CCS (8 digits).

xorData

If the formatted PIN is encrypted twice to build the resulting PIN block, this data can be used to modify the result of the first encryption by an XOR-operation. This parameter is a string of hexadecimal data that must be converted by the client, e.g. 0x0123456789ABCDEF must be converted to 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x41 0x42 0x43 0x44 0x45 0x46 and terminated with 0x00. In other words the client would set xorData to "0123456789ABCDEF". The hex digits 0xA to 0xF can be represented by characters in the ranges 'a' to 'f' or 'A' to 'F'. If this value is NULL no XOR-operation will be performed. If the formatted PIN is not encrypted twice (i.e. if lpsKeyEncKey is NULL) this parameter is ignored.

padding

Specifies the padding character. The valid range is 0x0 to 0xF in hexadecimal string format. Only the least significant nibble is used. This field is ignored for PIN block formats with fixed, sequential or random padding.

format

Specifies the format of the PIN block. Possible values are: (see command [Crypto.Capabilities](#))

- ibm3624 - PIN left justified, filled with padding characters, PIN length 4-16 digits. The padding character is a hexadecimal digit in the range 0x00 to 0x0F."
- ansi - PIN is preceded by 0x00 and the length of the PIN (0x04 to 0x0C), filled with padding character 0x0F to the right, PIN length 4-12 digits, XORed with PAN (Primary Account Number, minimum 12 digits without check number).
- iso0 - PIN is preceded by 0x00 and the length of the PIN (0x04 to 0x0C), filled with padding character 0x0F to the right, PIN length 4-12 digits, XORed with PAN (Primary Account Number without check number, no minimum length specified, missing digits are filled with 0x00).
- iso1 - PIN is preceded by 0x01 and the length of the PIN (0x04 to 0x0C), padding characters are taken from a transaction field (10 digits).
- eci2 - PIN left justified, filled with padding characters, PIN only 4 digits.
- eci3 - PIN is preceded by the length (digit), PIN length 4-6 digits, the padding character can range from 0x0 through 0xF.
- visa - PIN is preceded by the length (digit), PIN length 4-6 digits. If the PIN length is less than six digits the PIN is filled with 0x0 to the length of six, the padding character can range from 0x0 through 0x9 (This format is also referred to as VISA2).
- diebold - PIN is padded with the padding character and may be not encrypted, single encrypted or double encrypted.
- dieboldco - PIN with the length of 4 to 12 digits, each one with a value of 0x0 to 0x9, is preceded by the one-digit coordination number with a value from 0x0 to 0xF, padded with the padding character with a value from 0x0 to 0xF and may be not encrypted, single encrypted or double encrypted.
- visa3 - PIN with the length of 4 to 12 digits, each one with a value of 0x0 to 0x9, is followed by a delimiter with the value of 0xF and then padded by the padding character with a value between 0x0 to 0xF.
- banksys - PIN is encrypted and formatted according to the Banksys PIN block specifications.
- emv - The PIN block is constructed as follows: PIN is preceded by 0x02 and the length of the PIN (0x04 to 0x0C), filled with padding character 0x0F to the right, formatted up to 248 bytes of other data as defined within the EMV 4.0 specifications and finally encrypted with an RSA key.
- iso3 - PIN is preceded by 0x03 and the length of the PIN (0x04 to 0x0C), padding characters sequentially or randomly chosen, XORed with digits from PAN.
- ap - PIN is formatted according to the Italian Bancomat specifications. It is known as the Authentication Parameter PIN block and is created with a 5 digit PIN, an 18 digit PAN, and the 8 digit CCS from the track data.

key

Specifies the key used to encrypt the formatted PIN for the first time, this field is not required if no encryption is required. If this specifies a double-length or triple-length key, triple DES encryption will be performed. The key referenced by key property must have the function or pinRemote attribute. If this specifies an RSA key, RSA encryption will be performed

secondEncKey

Specifies the key used to format the once encrypted formatted PIN, this field is not required if no second encryption required. The key referenced by lpsKeyEncKey must have the function or pinRemote attribute. If this specifies a double-length or triple-length key, triple DES encryption will be performed.

pinBlockAttributes

This parameter specifies the encryption algorithm, cryptographic method, and mode to be used for this command. For a list of valid values see the [Capabilities.pinBlockAttributes](#) field. For a list of valid values see the [Capabilities.cryptAttributes](#) capability field. The values specified must be compatible with the key identified by key.

pinBlockAttributes/algorithm

Specifies the encryption algorithms supported by the PinBlock command. The following values are possible:

- A - AES.
- D - DEA.
- R - RSA.
- T - Triple DEA (also referred to as TDEA).

pinBlockAttributes/cryptoMethod

This parameter specifies the cryptographic method that will be used with the encryption algorithm specified by algorithm. If algorithm is 'A', 'D', or 'T', then this property cryptoMethod can be one of the following values:"

- ecb - The ECB encryption method.
- cbc - The CBC encryption method.
- cfb - The CFB encryption method.
- ofb - The OFB encryption method.
- ctr - The CTR method defined in NIST SP800-38A.

- xts - The XTS method defined in NIST SP800-38E.

If algorithm is 'R', then this property cryptoMethod can be one of the following values:

- rsaesPkcs1V15 - Use the RSAES_PKCS1-v1.5 algorithm.
- rsaesOaep - Use the RSAES OAEP algorithm.

Completion Message

| Payload | Type | Required |
|---|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

Event Messages

None

[10.3.8 - Pinpad.GetPinBlock](#)

This function takes the account information and a PIN entered by the user to build a formatted PIN. Encrypting this formatted PIN once or twice returns a PIN block which can be written on a magnetic card or sent to a host. The PIN block can be calculated using one of the algorithms specified in the [Pinpad.Capabilities](#) command. This command will clear the PIN unless the client has requested that the PIN be maintained through the [Pinpad.MaintainPin](#) command.

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Command Message

| Payload | Type | Required |
|---|---------|----------|
| { | | |
| " customerData ": Add example to YAML, | string | |
| " xorData ": Add example to YAML, | string | |
| " padding<td>integer</td><td></td> | integer | |
| " format ": "ibm3624", | string | |
| " key ": Add example to YAML, | string | |
| " secondEncKey ": Add example to YAML, | string | |
| " pinBlockAttributes ": { | object | |
| " algorithm ": "A", | string | |
| " cryptoMethod ": "ecb" | string | |
| } | | |
| } | | |

Properties**customerData**

The customer data should be an ASCII string. Used for ANSI, ISO-0 and ISO-1 algorithm to build the formatted PIN. For ANSI and ISO-0 the PAN (Primary Account Number, without the check number) is supplied, for ISO-1 a ten digit transaction field is required. If not used a NULL is required. Used for DIBOLD with coordination number, as a two digit coordination number. Used for EMV with challenge number (8 bytes) coming from the chip card. This number is passed as unpacked string, for example: 0123456789ABCDEF = 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x41 0x42 0x43 0x44 0x45 0x46 For AP PIN blocks, the data must be a concatenation of the PAN (18 digits including the check digit), and the CCS (8 digits).

xorData

If the formatted PIN is encrypted twice to build the resulting PIN block, this data can be used to modify the result of the first encryption by an XOR-operation. This parameter is a string of hexadecimal data that must be converted by the client, e.g. 0x0123456789ABCDEF must be converted to 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x41 0x42

0x43 0x44 0x45 0x46 and terminated with 0x00. In other words the client would set xorData to “0123456789ABCDEF”. The hex digits 0xA to 0xF can be represented by characters in the ranges ‘a’ to ‘f’ or ‘A’ to ‘F’. If this value is NULL no XOR-operation will be performed. If the formatted PIN is not encrypted twice (i.e. if lpsKeyEncKey is NULL) this parameter is ignored.

padding

Specifies the padding character. The valid range is 0x0 to 0xF in hexadecimal string format. Only the least significant nibble is used. This field is ignored for PIN block formats with fixed, sequential or random padding.

format

Specifies the format of the PIN block. Possible values are: (see command [Crypto.Capabilities](#))

- ibm3624 - PIN left justified, filled with padding characters, PIN length 4-16 digits. The padding character is a hexadecimal digit in the range 0x00 to 0x0F."
- ansi - PIN is preceded by 0x00 and the length of the PIN (0x04 to 0x0C), filled with padding character 0x0F to the right, PIN length 4-12 digits, XORed with PAN (Primary Account Number, minimum 12 digits without check number).
- iso0 - PIN is preceded by 0x00 and the length of the PIN (0x04 to 0x0C), filled with padding character 0x0F to the right, PIN length 4-12 digits, XORed with PAN (Primary Account Number without check number, no minimum length specified, missing digits are filled with 0x00).
- iso1 - PIN is preceded by 0x01 and the length of the PIN (0x04 to 0x0C), padding characters are taken from a transaction field (10 digits).
- eci2 - PIN left justified, filled with padding characters, PIN only 4 digits.
- eci3 - PIN is preceded by the length (digit), PIN length 4-6 digits, the padding character can range from 0x0 through 0xF.
- visa - PIN is preceded by the length (digit), PIN length 4-6 digits. If the PIN length is less than six digits the PIN is filled with 0x0 to the length of six, the padding character can range from 0x0 through 0x9 (This format is also referred to as VISA2).
- diebold - PIN is padded with the padding character and may be not encrypted, single encrypted or double encrypted.
- dieboldco - PIN with the length of 4 to 12 digits, each one with a value of 0x0 to 0x9, is preceded by the one-digit coordination number with a value from 0x0 to 0xF, padded with the padding character with a value from 0x0 to 0xF and may be not encrypted, single encrypted or double encrypted.
- visa3 - PIN with the length of 4 to 12 digits, each one with a value of 0x0 to 0x9, is followed by a delimiter with the value of 0xF and then padded by the padding character with a value between 0x0 to 0xF.

- banksys - PIN is encrypted and formatted according to the Banksys PIN block specifications.
- emv - The PIN block is constructed as follows: PIN is preceded by 0x02 and the length of the PIN (0x04 to 0x0C), filled with padding character 0x0F to the right, formatted up to 248 bytes of other data as defined within the EMV 4.0 specifications and finally encrypted with an RSA key.
- iso3 - PIN is preceded by 0x03 and the length of the PIN (0x04 to 0x0C), padding characters sequentially or randomly chosen, XORed with digits from PAN.
- ap - PIN is formatted according to the Italian Bancomat specifications. It is known as the Authentication Parameter PIN block and is created with a 5 digit PIN, an 18 digit PAN, and the 8 digit CCS from the track data.

key

Specifies the key used to encrypt the formatted PIN for the first time, this field is not required if no encryption is required. If this specifies a double-length or triple-length key, triple DES encryption will be performed. The key referenced by key property must have the function or pinRemote attribute. If this specifies an RSA key, RSA encryption will be performed

secondEncKey

Specifies the key used to format the once encrypted formatted PIN, this field is not required if no second encryption required. The key referenced by lpsKeyEncKey must have the function or pinRemote attribute. If this specifies a double-length or triple-length key, triple DES encryption will be performed.

pinBlockAttributes

This parameter specifies the encryption algorithm, cryptographic method, and mode to be used for this command. For a list of valid values see the [Capabilities.pinBlockAttributes](#) field. For a list of valid values see the [Capabilities.cryptAttributes](#) capability field. The values specified must be compatible with the key identified by key.

pinBlockAttributes/algorithm

Specifies the encryption algorithms supported by the PinBlock command. The following values are possible:

- A - AES.
- D - DEA.
- R - RSA.
- T - Triple DEA (also referred to as TDEA).

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

pinBlockAttributes/cryptoMethod

This parameter specifies the cryptographic method that will be used with the encryption algorithm specified by algorithm. If algorithm is 'A', 'D', or 'T', then this property cryptoMethod can be one of the following values:"

- ecb - The ECB encryption method.
- cbc - The CBC encryption method.
- cfb - The CFB encryption method.
- ofb - The OFB encryption method.
- ctr - The CTR method defined in NIST SP800-38A.
- xts - The XTS method defined in NIST SP800-38E.

If algorithm is 'R', then this property cryptoMethod can be one of the following values:

- rsaesPkcs1V15 - Use the RSAES_PKCS1-v1.5 algorithm.
- rsaesOaep - Use the RSAES OAEP algorithm.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "keyNotFound", | string | |
| "pinBlock": Add example to YAML | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- keyNotFound - The specified key was not found.
- accessDenied - The encryption module is either not initialized or not ready for any vendor specific reason.
- keyNoValue - The specified key name was found but the corresponding key value has not been loaded.
- useViolation - The use specified by keyUsage is not supported.
- noPin - The PIN has not been entered was not long enough or has been cleared.
- formatNotSupported - The specified format is not supported.
- invalidKeyLength - The length of secondEncKey or key is not supported by this key or the length of an encryption key is not compatible with the encryption operation required.
- algorithmNotSupported - The algorithm specified by algorithm is not supported.
- dukptOverflow - The DUKPT KSN encryption counter has overflowed to zero. A new IPEK must be loaded.
- modeNotSupported - The mode specified by bModeOfUse is not supported
- cryptoMethodNotSupported - The cryptographic method specified by cryptoMethod is not supported.

pinBlock

The encrypted PIN block formatted in base64

Event Messages

- [Pinpad.DUKPTKSNEvent](#)

10.4 - Event Messages

10.4.1 - [Pinpad.DUKPTKSNEvent](#)

This event sends the DUKPT KSN of the key used in the command. The receiving TRSM uses this to derive the key from the BDK.

| Payload | Type | Required |
|---------|------|----------|
| { | | |

```
"key": Add example to YAML, string  
"ksn": Add example to YAML string  
}
```

Properties

key

Specifies the name of the DUKPT Key derivation key.

ksn

structure that contains the KSN formatted in base64.

10.5 - Unsolicited Messages

10.5.1 - Pinpad.IllegalKeyAccessEvent

This event specifies that an error occurred accessing an encryption key. Possible situations for generating this event are listed in the description of lErrorCode.

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "keyName": Add example to YAML, string | | |
| "errorCode": "keynotfound" | string | |
| } | | |

Properties

keyName

Specifies the name of the key that caused the error.

errorCode

Specifies the type of illegal key access that occurred

11 - Printer Interface

This chapter defines the Printer interface functionality and messages.

11.1 - Summary

This specification describes the functionality of the services provided by banking printers and scanning devices under XFS, focusing on the following areas:

- client programming for printing
- print document definition
- scanning images for devices such as check scanners

The XFS printer interface is implemented around a forms model which also standardizes the basic document definition.

11.2 - General Information

11.2.1 - Banking Printer Types

The XFS printer service defines and supports five types of banking printers through a common interface:

- **Receipt Printer**

The receipt printer is used to print cut sheet documents. It may or may not require insert or eject operations, and often includes an operator identification device, e.g. Teller A and Teller B lights, for shared operation.

- **Journal Printer**

The journal is a continuous form device used to record a hardcopy audit trail of transactions, and for certain report printing requirements.

- **Passbook Printer**

The passbook device is physically and functionally the most complex printer. The XFS definition supports automatic positioning of the book, as well as read/write capability for an optional integrated magnetic stripe. The implementation also manages the book geometry - i.e. the margins and centerfolds - presenting the simplest possible client interface while delivering the full range of functionality.

Some passbook devices also support the dispensing of new passbooks from up to four passbook paper sources (upper, aux, aux2, lower). Some passbook devices may also be able to place a full passbook in a parking station, print the new passbook and

return both to the customer. Passbooks can only be dispensed or moved from the parking station if there is no other media in the print position or in the entry/exit slot.

- **Document Printer**

Document printing is similar to receipt printing - a set of fields are positioned on one or more inserted sheets of paper - but the focus is on full-size forms. It should be noted that the XFS environment supports the printing of text and graphic fields from the client. The electronic printing of the form image (the template portion of the form which is usually pre-printed with dot-matrix style printers) may also be printed by the client.

- **Scanner Printer**

The scanner printer is a device incorporating both the capabilities to scan inserted documents and optionally to print on them. These devices may have more than one area where documents may be retained.

Additional hardware components, like scanners, stripe readers, OCR readers, and stamps, normally attached directly to the printer are also controlled through this interface.

Additionally, the Printer and Scanning class interface can also be used for devices that are capable of scanning without necessarily printing. This includes devices such as Check Scanners.

The specification refers to the terms paper and media. When the term paper is used this refers to paper that is situated in a paper supply attached to the device. The term media is used for media that is inserted by the customer (e.g. check and other material that is scanned) or that is issued to the customer (e.g. a receipt or statement). Receipt, document and passbook printers with white passbook dispensing capability have both. As soon as the paper gets printed it becomes media. Scanners only have media. The term media does not apply to journal printers. When paper is in the print position it is classified as media, on some printers that maintain paper under the print head there will always be both media and paper.

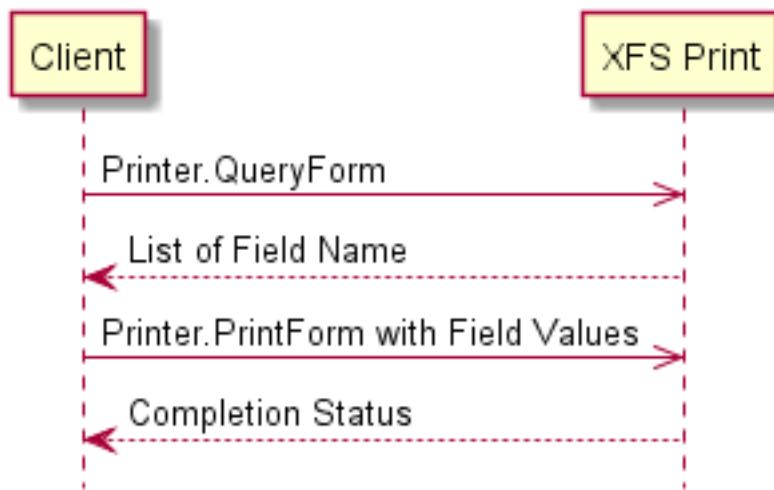
11.2.2 - Forms Model

The XFS printing class functionality is based on a “forms” model for printing. Banking documents are represented as a series of text and/or graphic fields output from the client and positioned on the document by the XFS printing system.

The form is an object which includes the positioning and presentation information for each of the fields in the document. The client selects a form and supplies only the field data and the control parameters to fully define the print document.

The form objects are owned and managed by the XFS printing service. To optimize maintainability of the system, the client can query the service for the list of fields required to print a given form. Through this mechanism, it is not necessary to duplicate the field

contents of forms in client authoring data. The figure below outlines the printing process from the client's view.



The XFS implementation recognizes that the form object must be supported by job-specific data to fully address printing requirements. As an example, a form defining a passbook print line will need to have its origin defined externally in order to be reused for different passbook lines. These job specific parameters are supplied on the [Printer.PrintForm](#) command.

In some cases, the client wants to print a block of data without considering it as a series of separate fields. One example is a line of journal data, fully formatted by the client. This can be handled by defining a one field form, or by use of the [Printer.RawData](#) command.

The document definition under XFS printing is standardized to provide portability across vendor implementations. The standard has been defined at the source language level for the document definition, allowing vendor differences at the runtime level to manage implementation specific dependencies, providing several areas where vendors can provide value-added extensions. As an example, a vendor providing a graphical form definition tool can produce the field definition object format directly. The XFS requirements for portability are:

- A vendor must be able to export print format in the standardized field definition source format for portability to other systems.
- A vendor must be able to import document formats produced on other systems in the standardized field definition source format.
- A vendor can extend the field definition source language, but any verbs included in the standard must be implemented strictly as defined by the standard. Import and export facilities must be tolerant of source language extensions, reporting but ignoring the exceptions.

11.2.3 - Command Overview

The basic operation of the print devices is managed using the two primary commands:

- **Printer.GetQueryForm**

This command retrieves the form header information, and the list of fields.

- **Printer.PrintForm**

This command includes as parameter data the name of the form to select and the required field data values.

This approach combines in the most efficient manner the four logical steps required to print a form:

- Selecting a document form object.
- Querying the service for the list of fields.
- Supplying the data for each field.
- Issuing the print command.

By using *Printer.GetQueryForm* command to retrieve the list of field names, it is possible for a client to assemble the required set of fields for a form before locking the service. This minimizes the time that each client request ties up the service. Using *Printer.GetQueryForm*, it is also possible to query the attributes of a field. This command is generally not required for most clients.

The combination of form selection, field value presentation and the print action make it possible to express a complete print operation with *Printer.PrintForm* command. Where these multiple print functions represent a series of passbook lines (using the INDEX capability in the field definition), the *Printer.PrintForm* command provides support for management of the print line number. Note that if a form contains a tabular field (i.e. one with a non-zero INDEX value), and data is not supplied for some of the lines in the “table”, then those lines are left blank.

For printers with the capability to read from a passbook (OCR, MICR and/or magnetic stripe), the data is read with the *Printer.ReadForm* command. The data is written using the *Printer.PrintForm* command. Since these devices are usable only for passbook operations, they are not defined as separate logical devices.

Finally, the *Printer.PrintRawFile* command can be used to print a file that contains a complete print job in the native printer language. This file will have been created using the native Operating System API (e.g. Windows GDI).

11.2.4 - Form, Sub-Form, Field, Frame, Table and Media Definitions

This section outlines the format of the definitions of forms, the fields within them, optional tables and fields within the form, and the media on which they are printed.

11.2.4.1 - Definition Syntax

The syntactic rules for form, field and media definitions are as follows:

- **White space**
space, tab
- **Line continuation**
backslash (\)
- **Line termination**
CR, LF, CR/LF; line termination ends a "keyword section" (a keyword and its value[s])
- **Keywords**
must be all upper case
- **Names**
(field/media/font names) any case; case is preserved;

Service Providers are case sensitive

- **Strings**
all strings must be enclosed in double quote characters (");
standard C escape sequences are allowed.
- **Comments**
start with two forward slashes (//), end at line termination

Other Notes:

- The values of a keyword are separated by commas.
- If a keyword is present, all its values must be specified; default values are used only if the keyword is absent.
- Values that are character strings are marked with * in the definitions below and must be quoted as specified above.
- The order of attributes within the forms is not mandatory and the attributes may be defined in any order.
- All forms can be represented using either ISO 646 (ANSI) or UNICODE character encoding. If the UNICODE representation is used, then all Names and Strings are restricted to an internal representation of ISO 646 (ANSI) characters. Only the INITIALVALUE and FORMAT keyword values can have double byte values outside of the ISO 646 (ANSI) character set.
- If forms character encoding is UNICODE then, consistent with the UNICODE standard, the file prefix must be in Little Endian (0xFFFFE) or Big Endian (0xFEFF) notation, such that UNICODE encoding is recognized.
- A form and its optional subforms that have multiple XFSFIELDS with the same fieldname are invalid. The *formInvalid* error will be returned if specified in the input to the command.
- A form that has multiple XFSSUBFORMS with the same subformname is invalid. The *formInvalid* error will be returned if specified in the input to the command.

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

- A form and its optional subforms that have multiple XFSFRAMES with the same framename are invalid. The formInvalid error will be returned if specified in the input to the command.

11.2.4.2 - Form and Media Measurements

The UNIT keyword sections of the form and media definitions specify the base horizontal and vertical resolution as follows:

- The *base* value specifies the base unit of measurement.
- The *x* and *y* values specify the horizontal and vertical resolution as fractions of the base value (e.g. an *x* value of 10 and a base value of MM means that the base horizontal resolution is 0.1 mm).

The base resolutions thus defined by the UNIT keyword section of the XFSFORM definition are used as the units of the form definition keyword sections:

- SIZE (*width* and *height* values)
- ALIGNMENT (*xoffset* and *yoffset* values)

and of the sub-form definition keyword sections:

- POSITION (*x* and *y* values)
- SIZE (*width* and *height* values)

and of the field definition keyword sections:

- POSITION (*x* and *y* values)
- SIZE (*width* and *height* values)
- INDEX (*xoffset* and *yoffset* values)

and of the frame definition keyword sections:

- POSITION (*x* and *y* values)
- SIZE (*width* and *height* values)
- REPEATONX (*xoffset* value)
- REPEATONY (*yoffset* value)

The base resolutions thus defined by the UNIT keyword section of the XFSMEDIA definition are used as the units of the media definition keyword sections:

- SIZE (*width* and *height* values)
- PRINTAREA (*x*, *y*, *width* and *height* values)
- RESTRICTED (*x*, *y*, *width* and *height* values)

NOTE: The origin for coordinate based systems is (0,0). The origin for row/column based systems can be (0,0) or (1,1) and must be configurable within the Service Provider.

11.2.4.3 - Form Definition

Attributes are not required in any mandatory order within a Form definition.

XFSFORM

| | | |
|-----------------------------|-------------------|---|
| XFSFORM | <i>formname*</i> | |
| BEGIN | | |
| (required) UNIT | <i>base,</i> | Base resolution unit for form definition: MM INCH ROWCOLUMN |
| | <i>x,</i> | Horizontal base unit fraction |
| | <i>y</i> | Vertical base unit fraction |
| (required) SIZE | <i>width,</i> | Width of form |
| | <i>height</i> | Height of form |
| (required) ALIGNMENT | <i>alignment,</i> | Alignment of the form on the physical media (TOPLEFT (default), TOPRIGHT, BOTTOMLEFT, BOTTOMRIGHT). This option allows the positioning of a form onto a physical page relative to any combination of the edges of the physical media, to support the variations in how devices sense the edge of page for positioning purposes. |
| | <i>xoffset,</i> | Horizontal offset relative to the horizontal alignment specified by alignment. Always specified as a positive value (i.e. if aligned to the right side of the media, means offset the form to the left). (default = 0) |
| | <i>yoffset</i> | Vertical offset relative to the vertical alignment specified by alignment. Always specified as a positive value (i.e. if aligned to the bottom of the media, means offset the form upward). (default = 0) |
| ORIENTATION | <i>type</i> | Orientation of form: PORTRAIT (default) LANDSCAPE |
| SKEW | <i>skewfactor</i> | Maximum skew factor in degrees (default = |

| | | |
|-----------------------------------|---------------------|---|
| | | 0) |
| VERSION | <i>major,</i> | Major version number |
| | <i>minor,</i> | Minor version number |
| | <i>date*</i> , | Creation/modification date |
| | <i>author*</i> | Author of the form |
| CPI | <i>cpi</i> | Characters per inch. This value specifies the default CPI within the form. When the ROWCOLUMN unit is used, the form CPI and LPI are used to calculate the position and size of all fields within a form, irrespective of the CPI and LPI of the fields themselves. |
| LPI | <i>lpi</i> | Lines per inch. This value specifies the default LPI within the form. When the ROWCOLUMN unit is used, the form CPI and LPI are used to calculate the position and size of all fields within a form, irrespective of the CPI and LPI of the fields themselves. |
| POINTSIZE | <i>pointsize</i> | This value specifies the default POINTSIZE within the form. |
| COPYRIGHT | <i>copyright*</i> | Copyright entry |
| TITLE | <i>title*</i> | Title of form |
| COMMENT | <i>comment*</i> | Comment section |
| USERPROMPT | <i>prompt*</i> | Prompt string for user interaction |
| [XFSFIELD BEGIN ... END] | <i>fieldname*</i> | One field definition (as defined in the next section) for each field in the form. The fieldname within a form and its optional subforms must be unique. |
| [XFSFRAME BEGIN ... END] | <i>framename*</i> | One frame definition (as defined in the next section) for each frame in the form. The framename within a form and its optional subforms must be unique. |
| [XFSSUBFORM BEGIN ... END] | <i>subformname*</i> | One subform definition (as defined in the next section) for each subform in the form. The subformname within a form must be unique. |
| END | | |

11.2.4.4 - SubForm Definition

XFSSUBFORM

| | | |
|-------------------|------------------------------------|--|
| XFSSUBFORM | | <i>subformname*</i> |
| BEGIN | | |
| (required) | POSITION | <i>x,</i> <i>y or (y, z)</i> |
| | | Horizontal position (relative to left side of form) Vertical position (relative to top of form). Format (y, z) is used to indicate vertical positioning relative to top of form when top of form is other than 1st page of form, where Z indicates page number (relative to 0) and Y indicates base resolution units relative to top of the form page number (as indicated by Z). Format y is used to indicate vertical positioning relative to top of the 1st form page. |
| (required) | SIZE | <i>width,</i> <i>height</i> |
| | | Width of subform. Width must not exceed width of form. Height of subform. Height must not exceed height of form. |
| | [XFSFIELD <i>fieldname*</i> | |
| | BEGIN | |
| | ... | |
| | END] | |
| | [XFSFRAME <i>framename*</i> | |
| | BEGIN | |
| | ... | |
| | END] | |
| END | | |

The XFSSUBFORM definition provides a means to isolate a selected area of a form where the user may want to have a select group of fields, frames, and/or running headers and footers. All field and frame definitions within a subform are relative to the POSITION of the subform. A form definition with an imbedded subform will have a series of statements illustrated as follows:

```

XFSFORM
BEGIN
*
*
XFSSUBFORM
BEGIN
  XFSFIELD
  BEGIN
    *
    *
  END
  XFSFIELD
  BEGIN
    *
    *
  END
END
END

```

11.2.4.5 - Field Definition

XFSFIELD

| | | |
|----------------------------|---|--|
| XFSFIELD | <i>fieldname</i> * | The fieldname within a form and its optional subforms must be unique. |
| BEGIN | | |
| (required) POSITION | <i>x</i> , <i>y</i> or (<i>y</i> , <i>z</i>) | Horizontal position (relative to left side of form/subform). Vertical position (relative to top of form/subform). Format (<i>y</i> , <i>z</i>) is used to indicate vertical positioning relative to top of form/subform when top of form/subform is other than 1st page of form/subform, where <i>z</i> indicates page number (relative to 0) and <i>y</i> indicates base resolution units relative to top of the form/subform page number (as indicated by <i>z</i>). Format <i>y</i> is used to indicate vertical positioning relative to top of the 1st form/subform. |
| FOLLOWS | <i>fieldname</i> * | Print this field directly following the field with the name <i>fieldname</i> ; positioning information is ignored. See the description of Printer.PrintForm . If FOLLOWS is omitted, then fields are printed in the order |

| | | |
|---------------|------------|---|
| HEADER | <i>N</i> | that they appear in the form definition. This field is either a form/subform header field. N represents a form/subform page number (relative to 0) the header field is to print within. |
| | <i>N-N</i> | N-N represents a form/subform page number range the header field is to print within. Combinations of N and N-N may exist separated by commas. |
| | <i>ALL</i> | ALL indicates that header field is to be printed on all pages of form/subform. The form/subform page number is intended to supplement the Z parameter of the POSITION keyword. For example, 0,2-4,6 indicates that the header field is to print on relative form/subform pages 0, 2, 3, 4, and 6. |
| HEADER | <i>N</i> | This field is either a form/subform header field. N represents a form/subform page number (relative to 0) the header field is to print within. |
| | <i>N-N</i> | N-N represents a form/subform page number range the header field is to print within. Combinations of N and N-N may exist separated by commas. |
| | <i>ALL</i> | ALL indicates that header field is to be printed on all pages of form/subform. The form/subform page number is intended to supplement the Z parameter of the POSITION keyword. For example, 0,2-4,6 indicates that the header field is to print on relative form/subform pages 0, 2, 3, 4, and 6. |
| FOOTER | <i>N</i> | This field is either a form/subform footer field. N represents a form/subform page number (relative to 0) the footer field is to print within. |
| | <i>N-N</i> | N-N represents a form/subform page number range the footer field is to print within. Combinations of N and N-N may exist separated by commas. |
| | <i>ALL</i> | ALL indicates that footer field is to be printed on all pages of form/subform. The |

| | | |
|-------------------|----------------|--|
| | | form/subform page number is intended to supplement the Z parameter of the POSITION keyword. For example, 0,2-4,6 indicates that the footer field is to print on relative form/subform pages 0, 2, 3, 4, and 6. |
| | SIDE | <i>side</i> Side of form where field is positioned: FRONT (default) BACK |
| (required) | SIZE | <i>width,</i> <i>height</i> Field width. Field height. |
| | INDEX | <i>repeatcount</i> Count how often this field is repeated in the form, INDEX fields are fixed length (default is no INDEX field). |
| | | <i>xoffset,</i> Horizontal offset for next field. |
| | | <i>yoffset</i> Vertical offset for next field. |
| | TYPE | <i>fieldtype</i> Type of field: TEXT (default) MICR OCR MSF BARCODE GRAPHIC PAGEMARK |
| | SCALING | <i>scalingtype</i> Information on how to size the GRAPHIC within GRAPHIC field types: BESTFIT (default): Scale to size indicated. ASIS: Render at native size. MAINTAINASPECT: scale as close as possible to size indicated while maintaining the aspect ratio and not losing graphic information. |
| | BARCODE | <i>hriposition</i> Position of the HRI (Human Readable Interpretation) characters: NONE (default) ABOVE BELOW BOTH |
| | | The type of barcode to print is defined in the FONT field. |

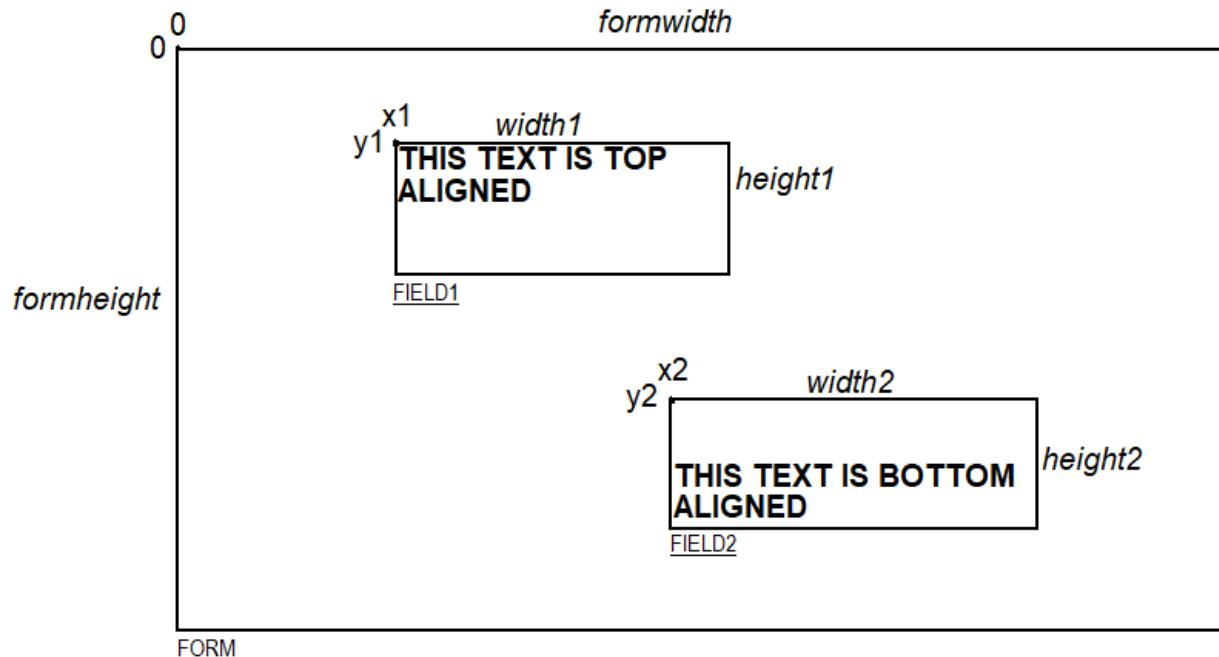
| | | |
|-------------------|-------------------|--|
| COERCIVITY | <i>coercivity</i> | Coercivity to be used for writing to the magnetic stripe of MSF field types: AUTO (default): Coercivity is decided by the Service Provider or hardware. LOW: Low coercivity. HIGH: High coercivity. |
| CLASS | <i>class</i> | Field class: OPTIONAL (default) STATIC REQUIRED |
| ACCESS | <i>access</i> | Access rights of field: WRITE (default) READ READWRITE |
| OVERFLOW | <i>overflow</i> | Action of field overflow: TERMINATE (default) TRUNCATE BESTFIT (The Service Provider fits the data into the field as well as it can) OVERWRITE (a contiguous write) WORDWRAP |
| STYLE | <i>style</i> | Display attributes as a combination, using the operator, of the following: NORMAL (default) BOLD ITALIC UNDER (single underline) DOUBLEUNDER (double underline) DOUBLE (double width) TRIPLE (triple width) QUADRUPLE (quadruple width) STRIKETHROUGH ROTATE90 (rotate 90 degrees clockwise) ROTATE270 (rotate 270 degrees clockwise) UPSIDEDOWN (upside down) PROPORTIONAL (proportional spacing) DOUBLEHIGH TRIPLEHIGH QUADRUPELHIGH CONDENSED SUPERSCRIPT OVERSCORE LETTERQUALITY |

| | | |
|-------------------|------------------|--|
| | | NEARLETTERQUALITY DOUBLESTRIKE OPAQUE (If omitted then the default attribute is transparent) |
| | | Some of these styles may be mutually exclusive or may combine to provide unexpected results. |
| CASE | <i>case</i> | Convert field contents to: NOCHANGE (default) UPPER LOWER |
| HORIZONTAL | <i>justify</i> | Horizontal alignment of field contents: LEFT (default) RIGHT CENTER JUSTIFY |
| VERTICAL | <i>justify</i> | Vertical alignment of field contents: BOTTOM (default) CENTER TOP |
| COLOR | <i>color</i> | Color name: BLACK (default) WHITE GRAY RED BLUE GREEN YELLOW |
| RGBCOLOR | <i>R, G, B</i> | Color in RGB 8 bits per color format: R: Red portion of the RGB value 0-255. G: Green portion of the RGB value 0-255. B: Blue portion of the RGB value 0-255. |
| FONT | <i>fontname*</i> | RGBCOLOR overrides the COLOR attribute. Font name: This attribute is interpreted by the Service Provider. In some cases, it may indicate printer resident fonts, and in others it may indicate the name of a downloadable font. For BARCODE fields it represents the barcode font name. In some cases, this pre-defines the following parameters: |
| POINTSIZE | <i>pointsize</i> | Point size. If unspecified, the point size defaults to the POINTSIZE defined for the |

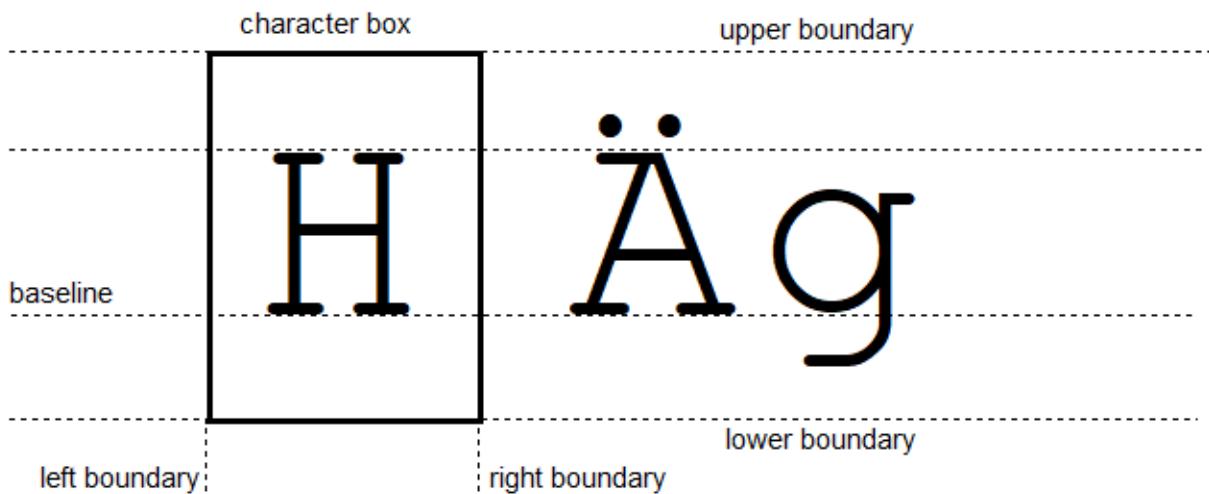
| | | |
|---------------------|----------------------|---|
| | | form. |
| CPI | <i>cpi</i> | Characters per inch. If unspecified, the CPI defaults to the CPI defined for the form. |
| LPI | <i>lpi</i> | Lines per inch. If unspecified, the LPI defaults to the LPI defined for the form. |
| FORMAT | <i>formatstring*</i> | This is a client defined input field describing how the client should format the data. This may be interpreted by the Service Provider. |
| INITIALVALUE | <i>value*</i> | Initial value. For GRAPHIC type fields, this value may contain the filename of the graphic image. The type of this graphic will be determined by the file extension (e.g. BMP for Windows Bitmap). Graphic file name may be full or partial path. For example, “C:\BSVC\BSVCLOGO.BMP” illustrates use of full path name. A file name specification of “LOGO.BMP” illustrates partial path name. In this instance file is obtained from current directory. Graphic contents can be changed dynamically at run-time and the new content will be printed on the next print action. |

END

The following diagrams illustrate the positioning and sizing of text fields on a form, and the vertical alignment of text within a field using **VERTICAL=TOP** and **VERTICAL=BOTTOM** values in the field definition.



Definition of the character drawing box:



When more than one line of text is to be printed in a field, and the definition includes **VERTICAL=BOTTOM**, the vertical position of the first line is calculated using the specified (or implied) **LPI** value.

[11.2.4.6 - Frame Definition](#)

XFSFRAME

XFSGRAME *framename**

BEGIN

| | | |
|----------------------------|--------------------|--|
| (required) POSITION | <i>X</i> | Horizontal position of top left corner of the frame (relative to left side of form/subform). |
| | <i>Y or (Y, Z)</i> | Vertical position of top left corner of frame (relative to top of form/subform). Format (Y, Z) is used to indicate vertical positioning of the top left corner of the frame relative to top of form/subform when top of form/subform is other than 1st page of form/subform, where Z indicates page number (relative to 0) and Y indicates base resolution units relative to top of the form/subform page number (as indicated by Z). Format Y is used to indicate vertical positioning of the left corner of frame relative to top of the 1st form/subform. |
| FRAMES | <i>fieldname*</i> | Frames the field with the name <i>fieldname</i> , positioning and size information are ignored. The frame surrounds the complete field, not just the printed data. If the field is repeated, the frame surrounds the first and last fields that are printed. |
| HEADER | <i>N</i> | This frame is either a form/subform header frame. N represents a form/subform page number (relative to 0) the header frame is to print within. |
| | <i>N-N</i> | N-N represents a form/subform page number range the header frame is to print within. Combinations of N and N-N may exist separated by commas. |
| | <i>ALL</i> | ALL indicates that header frame is to be printed on all pages of form/subform. The form/subform page number is intended to supplement the Z parameter of the POSITION keyword. For example, 0,2-4,6 indicates that the header frame is to print on relative form/subform pages 0, 2, 3, 4, and 6. |
| FOOTER | <i>N</i> | This frame is either a form/subform footer frame. N represents a form/subform page number (relative to 0) the footer frame is to print within. |
| | <i>N-N</i> | N-N represents a form/subform page number range the footer frame is to print |

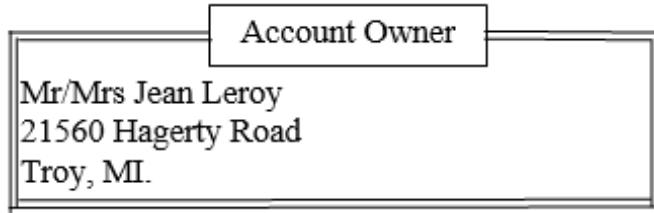
| | | |
|------------------------|---------------------|---|
| | | within. Combinations of N and N-N may exist separated by commas. |
| | <i>ALL</i> | ALL indicates that footer frame is to be printed on all pages of form/subform. The form/subform page number is intended to supplement the Z parameter of the POSITION keyword. For example, 0,2-4,6 indicates that the footer frame is to print on relative form/subform pages 0, 2, 3, 4, and 6. |
| SIDE | <i>side</i> | Side of form where this frame is positioned: FRONT (default) BACK |
| (required) SIZE | <i>width,</i> | Frame width in base horizontal units for the form. |
| | <i>height</i> | Frame height in base vertical units for the form. |
| REPEATONX | <i>repeatcount,</i> | Count how often this frame is repeated horizontally in the form. |
| | <i>xoffset</i> | Horizontal offset for next frame in base horizontal units. |
| REPEATONY | <i>repeatcount,</i> | Count how often this frame is repeated vertically in the form. |
| | <i>yoffset</i> | Vertical offset for next frame in base vertical units. |
| TYPE | <i>frametype</i> | Type of frame: RECTANGLE (default) ROUNDED_CORNER ELLIPSE |
| CLASS | <i>class</i> | Frame class: STATIC (default) OPTIONAL (The frame is printed only if its name appears in the list of field names given as parameter to the Printer.PrintForm command. In this case, the name of the frame must be different from all the names of the fields.) |
| OVERFLOW | <i>overflow</i> | Action on frame overflowing the form: TERMINATE (default) TRUNCATE BESTFIT (the Service Provider fits the frame into the media as well as it can) |

| | | |
|---------------------|----------------|---|
| STYLE | <i>style</i> | Frame line attributes: SINGLE_THIN (default) DOUBLE_THIN SINGLE_THICK DOUBLE_THICK DOTTED |
| COLOR | <i>color</i> | Color name for frame lines: BLACK (default) WHITE GRAY RED BLUE GREEN YELLOW |
| RGBCOLOR | <i>R, G, B</i> | Color in RGB 8 bits per color format: R: Red portion of the RGB value 0-255. G: Green portion of the RGB value 0-255. B: Blue portion of the RGB value 0-255. |
| | | RGBCOLOR overrides the COLOR attribute. |
| FILLCOLOR | <i>color</i> | Color name for interior of frame: BLACK WHITE (default) GRAY RED BLUE GREEN YELLOW |
| RGBFILLCOLOR | <i>R, G, B</i> | Color in RGB 8 bits per color format: R: Red portion of the RGB value 0-255. G: Green portion of the RGB value 0-255. B: Blue portion of the RGB value 0-255. |
| | | RGBFILLCOLOR overrides the FILLCOLOR attribute. |
| FILLSTYLE | <i>style</i> | Style for filling the interior of frame: NONE (default): No fill SOLID: Solid color BDIAGONAL: Downward hatch (left to right) at 45 degrees CROSS: Horizontal and vertical crosshatch DIAGCROSS: Crosshatch at 45 degrees FDIAGONAL: Upward hatch (left to right) at 45 degrees |

| | | |
|-------------------|------------------------|--|
| | | HORIZONTAL: Horizontal hatch VERTICAL: Vertical hatch |
| SUBSTSIGN | <i>substitute sign</i> | Character that is used as substitute sign when a character in a read field cannot be read |
| TITLE | <i>fieldname*</i> | Uses the field with the name as the title of the frame. Positioning information of the field is ignored. |
| HORIZONTAL | <i>justify</i> | Horizontal alignment of the frame title: LEFT (default) CENTER RIGHT |
| VERTICAL | <i>justify</i> | Vertical alignment of the frame title: TOP (default) BOTTOM |

END

The **XFSFRAME** definition provides a means for framing a **XFSFIELD** text field. The basic concept of a **XFSFRAME** definition and corresponding XFSFIELD definition is illustrated as follows:



When the **XFSFRAME** frames a field, its positioning and size information are ignored. Instead, Service Providers should position the top left corner of the frame one horizontal base unit to the left and one vertical base unit to the top of the top left corner of the field. Similarly, Service Providers should size the frame so that its bottom right corner is one base unit below and to the right of the field. For instance, if the form units are **ROWCOLUMN**, and a **XFSFRAME** "A" is said to frame the **XFSFIELD** "B" which is positioned at row 1, column 1 with a size of 1 row and 20 columns, the frame will be drawn from row 0, column 0 to row 3, column 22.

The horizontal and vertical positioning of a frame title overrides the position of the named **XFSFIELD**. For instance, if a **XFSFRAME** "A" is said to have the **XFSFIELD** "B" as its title, with the default horizontal and vertical title justification, it is just as if **XFSFIELD** "B" had been positioned at the top left corner of the frame. Note that the **SIZE** information for the title field still is meaningful; it gives the starting and/or ending positions of the frame lines.

The **SIDE** attributes of the **XFSFRAME** and the **XFSFIELDS** it refers to must agree.

The width of the lines and the interval between the lines of doubled frames are vendor specific. Whether the lines are drawn using graphics printing or using pseudo-graphic is vendor specific. However, Service Providers are responsible for rendering intersecting frames.

Depending on the printer technology, framing of fields can substantially slow down the print process.

Support of framing by a Service Provider or the device it controls is not mandatory to be XFS compliant.

The form:

```
XFSFORM "Multiple Balances"
BEGIN
UNIT INCH, 16, 16
SIZE 91, 64
VERSION 1, 0, "13/09/96", "XFS"
LANGUAGE 0x0409
XFSFIELD "Account Title"
BEGIN
POSITION 15, 4
SIZE 30, 4
CLASS STATIC
HORIZONTAL CENTER
INITIALVALUE "Account"
END
XFSFIELD "Balance Title"
BEGIN
POSITION 45, 4
SIZE 30, 4
CLASS STATIC
HORIZONTAL CENTER
INITIALVALUE "Balance"
END

XFSFIELD "Account"
BEGIN
POSITION 15, 8
SIZE 30, 4
INDEX 10, 0, 3
END // "Account"
XFSFIELD "Balance"
BEGIN
POSITION 45, 8
SIZE 30, 4
INDEX 10, 0, 3
HORIZONTAL RIGHT
END // "Balance"
XFSFRAME "Account Title"
BEGIN
POSITION 15, 4
FRAMES "Account Title"
SIZE 30, 4
STYLE DOUBLE_THIN
END
XFSFRAME "Balance Title"
BEGIN
POSITION 45, 4
```

```

FRAMES "Balance Title"
SIZE 30, 4
STYLE DOUBLE_THIN
END
XFSFRAME "Account"
BEGIN
  POSITION 15, 8
  FRAMES "Account"
  SIZE 30, 34
  STYLE DOUBLE_THIN
END
XFSFRAME "Balance"
BEGIN
  POSITION 45, 8
  FRAMES "Balance"
  SIZE 30, 34
  STYLE DOUBLE_THIN
END
END

```

When printed with the following field list:

```

Account[0]=0123456789123001
Account[1]=0123456789123002
Account[2]=0123456789123003
Balance[0]=$17465.12
Balance[1]=$2458.23
Balance[2]=$6542.78

```

Will print:

| Account | Balance |
|-----------------|------------|
| 012345678912300 | \$17465.12 |
| 1 | |
| 012345678912300 | \$2458.23 |
| 2 | |
| 012345678912300 | \$6542.78 |
| 3 | |

The form:

```
XFSFORM "Bank Details"
BEGIN
UNIT INCH, 16, 16
SIZE 121, 64
VERSION 1, 0, "13/09/96", "XFS Editor"
LANGUAGE 0x0409
XFSFIELD "Owner Frame Title"
BEGIN
    POSITION 24, 9

    SIZE 27, 3
    CLASS STATIC
    HORIZONTAL CENTER
    VERTICAL CENTER

    INITIALVALUE "Account Owner"
END
XFSFIELD "Owner"
BEGIN
    POSITION 20, 11
    SIZE 35, 9
    CLASS REQUIRED
    VERTICAL TOP
END // "Owner"
XFSFRAME "Owner Frame"
BEGIN
    POSITION 19, 10
    FRAMES "Owner"
    SIZE 37, 11
    TITLE "Owner Frame Title"
    HORIZONTAL CENTER
END
END
```

When printed with the following field list:

Owner = Mr./Mrs. Jean Leroy
21560 Hagerty Road
Troy, MI.

Will print:

| |
|--|
| Account Owner |
| Mr./Mrs. Jean Leroy 21560 Hagerty Road Troy, MI. |

The form:

```
XFSFORM "Bank Details"
BEGIN
UNIT INCH, 16, 16
SIZE 121, 64
VERSION 1, 0, "13/09/96", "XFS Editor"
LANGUAGE 0x0409
XFSFIELD "Owner"
BEGIN
POSITION 20, 11
SIZE 35, 9
CLASS REQUIRED
VERTICAL TOP
END
XFSFRAME "Owner Frame"
BEGIN
POSITION 19, 10
FRAMES "Owner"
SIZE 37, 11
FILLCOLOR GRAY
FILLSTYLE CROSS
END
END
```

When printed with the following field list:

```
Owner = Mr./Mrs. Jean Leroy
21560 Hagerty Road
Troy, MI.
```

Will print:

| | |
|---------------|--|
| Mr./Mrs. Jean | |
| Leroy | |
| 21560 Hagerty | |
| Road | |
| Troy, MI. | |

The form:

```
XFSFORM "Smart Account Number"
BEGIN
UNIT INCH, 16, 16
SIZE 121, 64
VERSION 1, 0, "13/09/96", "XFS Editor"
    LANGUAGE 0x0409
XFSFIELD "Account Number"
BEGIN
    POSITION 20, 8
    SIZE 4, 4
    INDEX 12, 4, 0
    HORIZONTAL CENTER
    VERTICAL CENTER
END
XFSFRAME "A/N Frame"
BEGIN
    POSITION 20, 8
    SIZE 4, 4
    REPEATONX 12, 4
    END
END
```

When printed with the following field list:

Account Number[0]=0
 Account Number[1]=1
 Account Number[2]=2
 Account Number[3]=3
 Account Number[4]=4
 Account Number[5]=5
 Account Number[6]=6
 Account Number[7]=7
 Account Number[8]=8
 Account Number[9]=9
 Account Number[10]=0
 Account Number[11]=1

Will print:

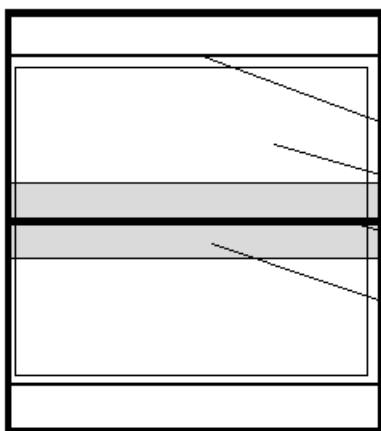
| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|

11.2.4.7 - Media Definition

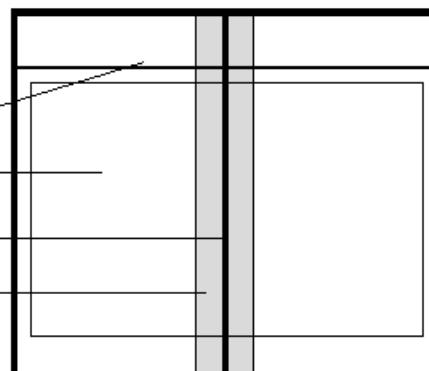
The media definition determines those characteristics that result from the combination of a particular media type together with a particular vendor's printer. The aim is to make it easy to move forms between different vendors' printers which might have different constraints on how they handle a specific media type. It is the Service Provider's responsibility to ensure that the form definition does not specify the printing of any fields that conflict with the media definition. An example of such a conflict might be that the form definition asks for a field to be printed in an area that the media definition defines as an unprintable area.

The media definition is also intended to provide the capability of defining media types that are specific to the financial industry. An example is a passbook as shown below.

Passbook with horizontal fold



Passbook with vertical fold



XFSMEDIA

| | | |
|------------------------|--------------------|---|
| XFSMEDIA | <i>medianame</i> * | |
| BEGIN | | |
| TYPE | <i>type</i> | Predefined media types are: GENERIC (default) MULTIPART PASSBOOK |
| SOURCE | <i>source</i> | Paper source: ANY (default) UPPER LOWER EXTERNAL (envelope tray or single sheet feed tray) AUX AUX2 PARK |
| (required) UNIT | <i>base</i> , | Base resolution unit for form definition: MM INCH ROWCOLUMN |
| | <i>x</i> , | Horizontal base unit fraction |
| | <i>y</i> | Vertical base unit fraction |
| (required) SIZE | <i>width</i> , | Width of physical media |
| | <i>height</i> | Height of physical media (0 = unlimited, i.e. roll paper) |
| PRINTAREA | <i>x</i> , | Printable area relative to top left corner of physical media (default = physical size of media) |
| | <i>y</i> , | |
| | <i>width</i> , | |
| | <i>height</i> | |
| RESTRICTED | <i>x</i> , | Restricted area relative to top left corner of physical media (default = no restricted area) |
| | <i>y</i> | |
| | <i>width</i> , | |
| | <i>height</i> | |

| | | |
|-------------------|-------------------|---|
| FOLD | <i>fold,</i> | Type of passbook: HORIZONTAL (default) VERTICAL |
| STAGGERING | <i>staggering</i> | Staggering of passbook from top (default = 0) |
| PAGE | <i>count</i> | Number of pages in passbook (default = 0) |
| LINES | <i>count</i> | Number of printable lines (default = 0) |

END

11.2.4.8 - XFS Form/Media Definition Files in Multi-Vendor Environments

Although for most Service Providers directory location and extension of XFS form/media definition files are configurable through the registry, the capabilities of Service Providers and or actual hardware may vary. Therefore, the following considerations should be taken into account when clients use XFS form definition files with the purpose of running in a multi-vendor environment:

- Physical print area dimensions of printers are not identical.
- Graphic printout may not be supported, which may limit the use of the FONT, CPI and LPI keywords.
- Some printers may have a resolution of dots/mm rather than dots/inch, which may result in printouts with a specific CPI/LPI font resolution to be slightly off size.
- Some form/media definition keywords may not be supported due to limitations of the hardware or software.

11.2.5 - Command and Event Flows during Single and Multi-Page / Wad Printing

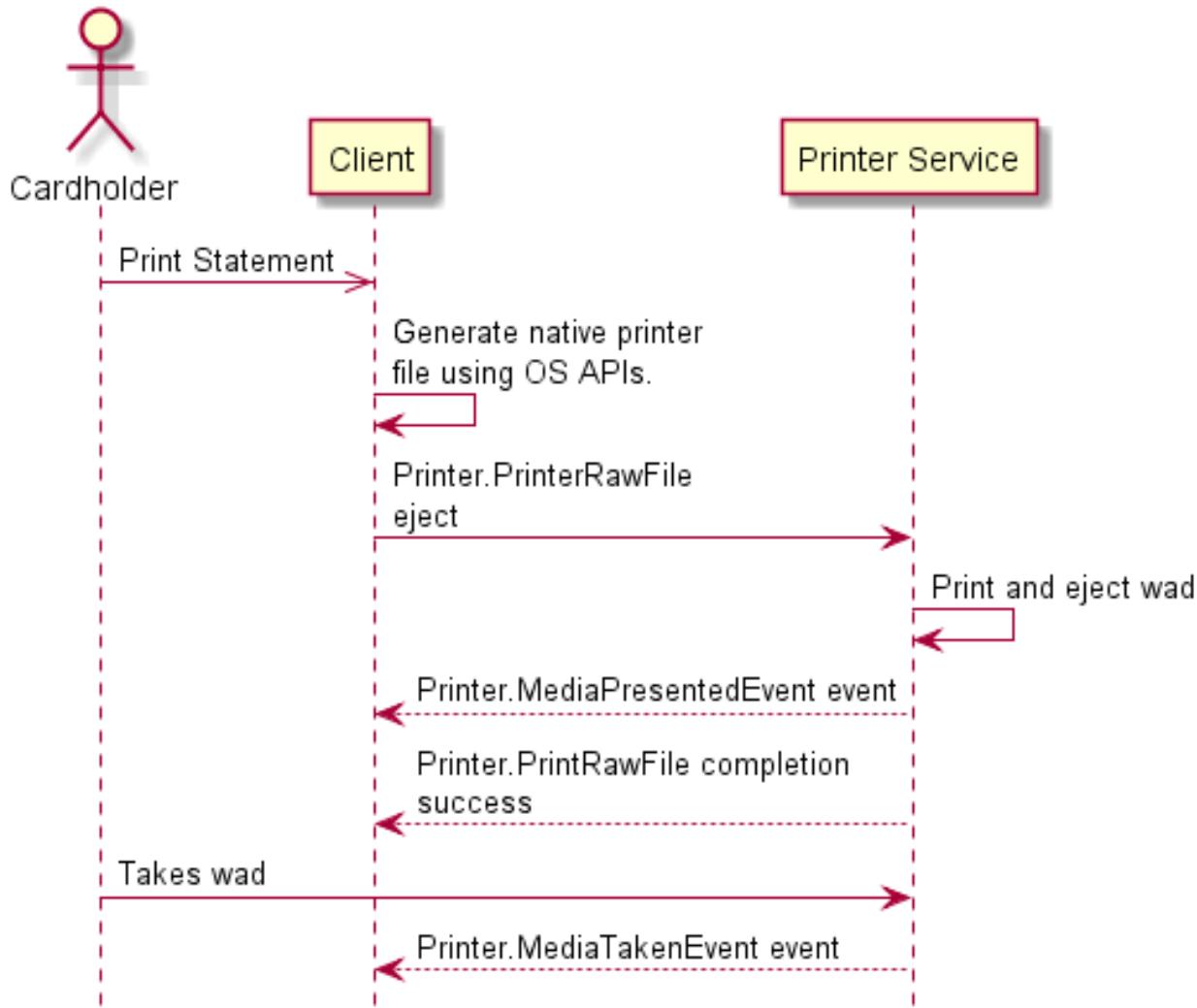
It is possible to print a number of pages or bunches of pages (wads) through the XFS Service Provider. The following sections describe how this is achieved.

11.2.5.1 - Single Page / Single Wad Printing With Immediate Media Control

This diagram illustrates the command and event flows in a successful print command (i.e. `Printer.PrintRawFile`, `Printer.PrintForm` and `Printer.RawData`) where the following conditions apply:

- A single page or single wad of pages is presented.

- The **mediaPresented** Capability flag is true (indicates that the **Printer.MediaPresentedEvent** event can be generated).
- The **mediaControl** flag in the command data is set to *eject*. The **Printer.PrintRawFile** command is used as an example.

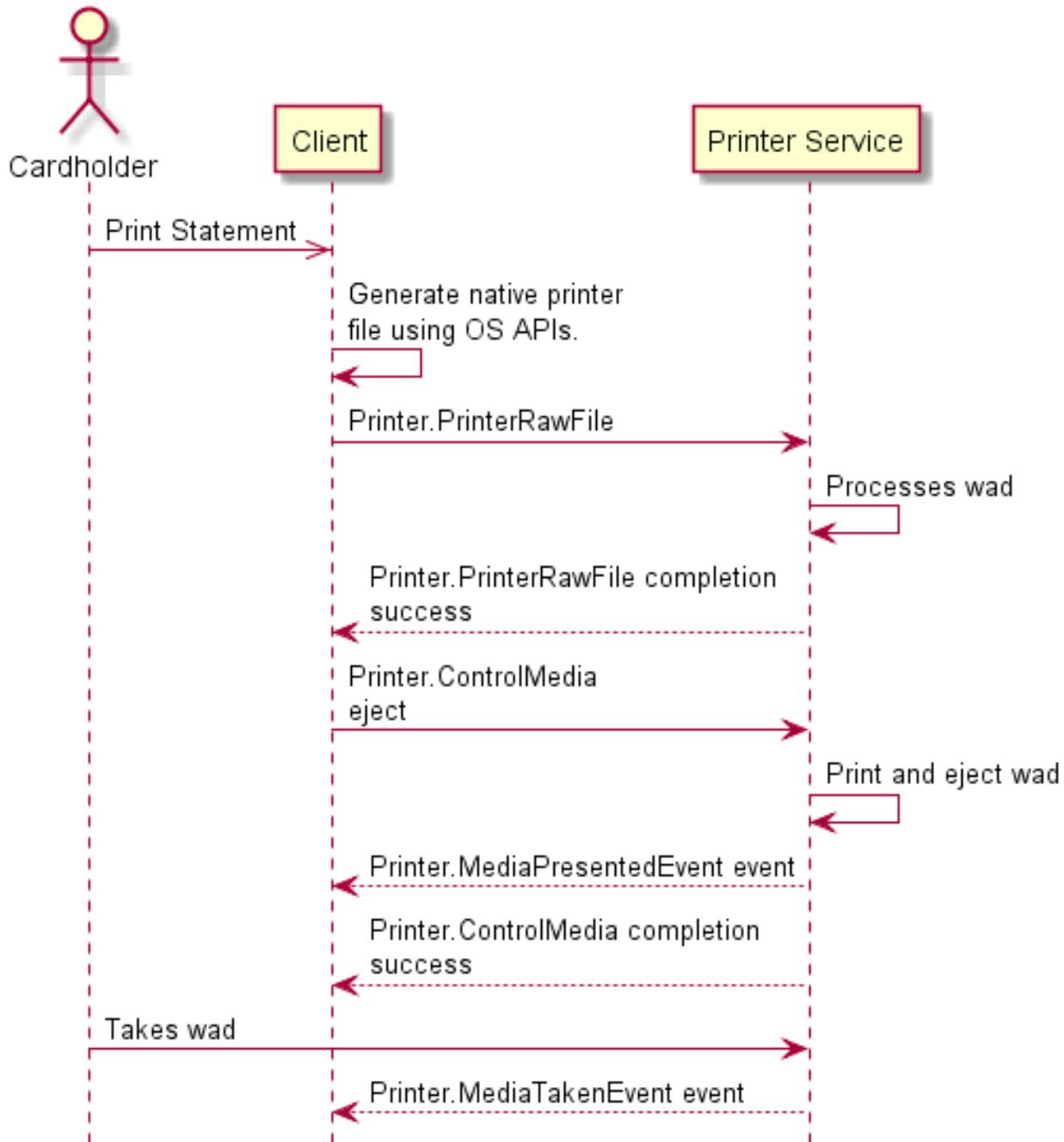


11.2.5.2 - Single Page / Single Wad Printing With Separate Media Control

This diagram illustrates the command and event flows in a successful print command (i.e. **Printer.PrintRawFile**, **Printer.PrintForm** and **Printer.RawData**) where the following conditions apply:

- A single page or single wad of pages is presented.
- The **mediaPresented** Capability flag is true (indicates that the **Printer.MediaPresentedEvent** event can be generated).

- The `mediaControl` flag is not included in the command data. The `Printer.PrintRawFile` command is used as an example.
- The media is presented to the user through a `Printer.ControlMedia` command, with the `mediaControl` parameter set to `eject`.



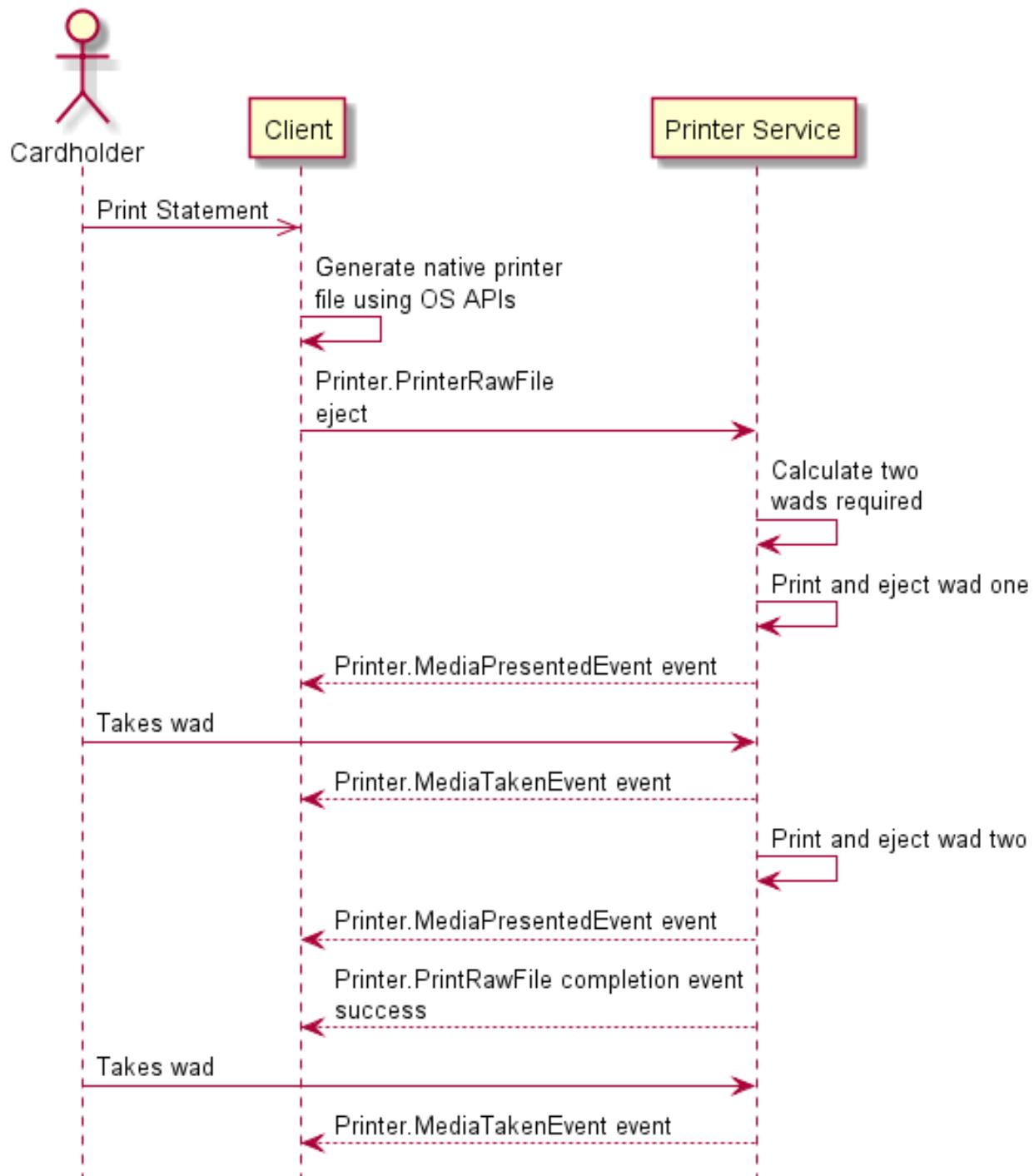
11.2.5.3 - Multi Page / Multi Wad Printing With Immediate Media Control

This diagram illustrates a successful `Printer.PrintRawFile` command where multiple page/wads are presented (and the `mediaPresented` capability indicates that the `Printer.MediaPresentedEvent` can be generated). In addition, the previous page/wad must be removed before subsequent pages/wads can be printed.

The diagram illustrates the command and event flows in a successful print command where the following conditions apply:

- Multiple pages or multiple wads of pages are presented.
- The `mediaPresented` capability is true (indicates that the `Printer.MediaPresentedEvent` event can be generated).
- The `mediaControl` flag in the command data is set to `eject`.
- The previous page/wad must be removed before subsequent pages/wads can be presented.

Note: work-in-progress. Use at your own risk.



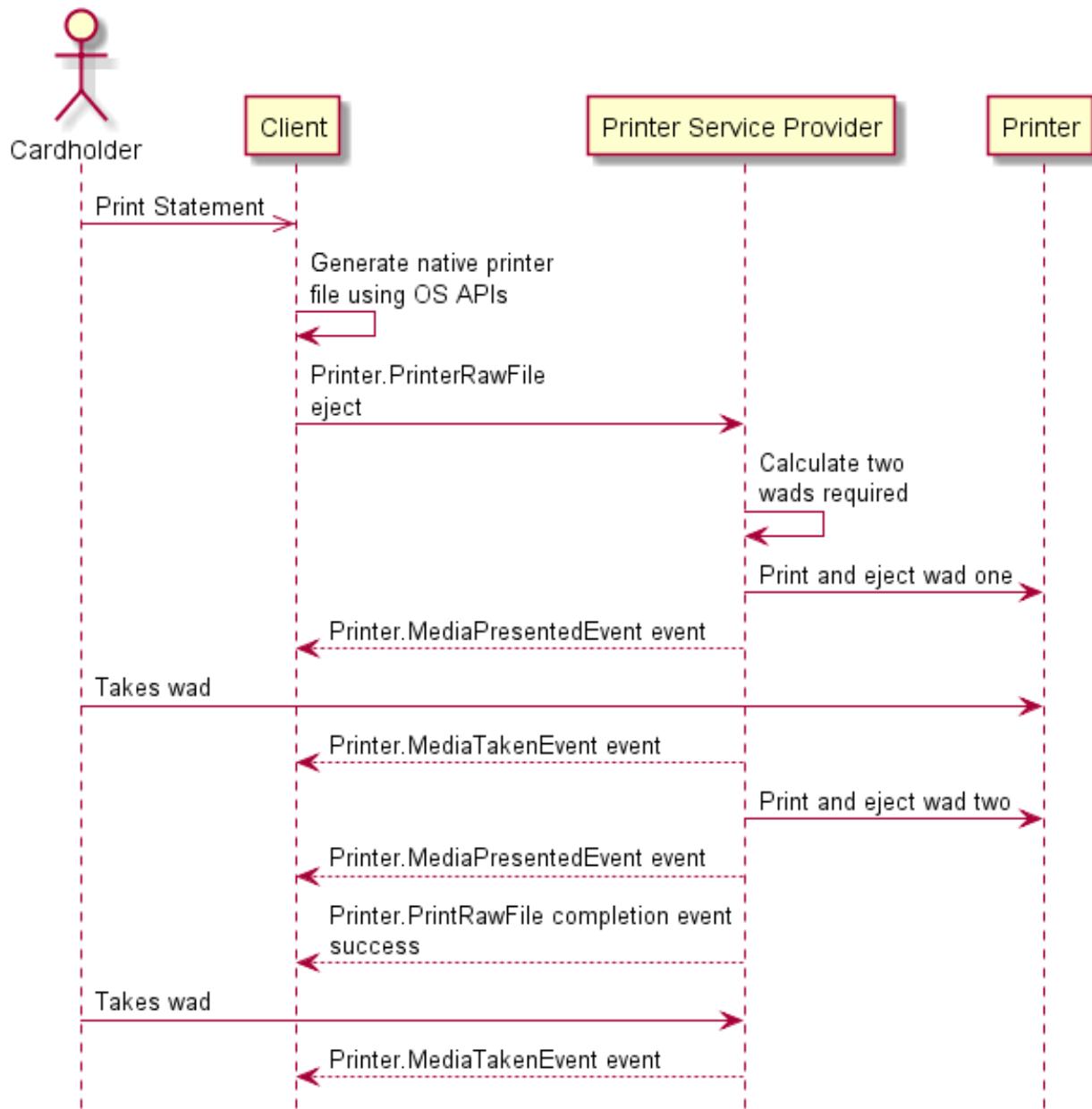
11.2.5.4 - Multi Page / Multi Wad Printing With Separate Media Control

This diagram illustrates a successful `Printer.PrintRawFile` command where multiple page / wads are presented (and the `mediaPresented` capability indicates that the

`Printer.MediaPresentedEvent` can be generated). In addition, the previous page/wad must be removed before subsequent pages/wads can be printed.

The diagram illustrates the command and event flows in a successful print command where the following conditions apply:

- Multiple pages or multiple wads of pages are presented.
- The *mediaPresented* capability is true (indicates that the `Printer.MediaPresentedEvent` event can be generated).
- The `mediaControl` flag command data is omitted.
- The media is presented to the user through a `Printer.ControlMedia` command, with the *mediaControl* property set to *eject*.
- The previous page/wad must be removed before subsequent pages/wads can be presented.



11.3 - Command Messages

11.3.1 - Printer.GetFormList

This command is used to retrieve the list of forms available on the device.

Command Message

| Payload | Type | Required |
|-----------------|---------|----------|
| { | | |
| "timeout": 5000 | integer | |
| } | | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "formList": [Add example to YAML] | array (string) | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

formList

The list of form names.

Event Messages

None

[11.3.2 - Printer.GetMediaList](#)

This command is used to retrieve the list of media definitions available on the device.

Command Message

| Payload | Type | Required |
|------------------------------------|------|----------|
| { "timeout": 5000 integer } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "mediaList": [Add example to YAML] | array (string) | |

}

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

mediaList

The list of media names.

Event Messages

None

11.3.3 - Printer.GetQueryForm

This command is used to retrieve details of the definition of a specified form.

Command Message

| Payload | Type | Required |
|---------------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "formName": Add example to YAML | string | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

formName

The form name for which to retrieve details.

Completion Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "formNotFound", | string | |
| "formName": Add example to YAML, | string | |
| "base": "inch", | string | |
| "unitX": 0, | integer | |
| "unitY": 0, | integer | |
| "width": 0, | integer | |
| "height": 0, | integer | |
| "alignment": "topLeft", | string | |
| "orientation": "portrait", | string | |
| "offsetX": 0, | integer | |
| "offsetY": 0, | integer | |
| "versionMajor": 0, | integer | |
| "versionMinor": 0, | integer | |
| "userPrompt": Add example to YAML, | string | |

```
"fields": [Add example to YAML]           array (string)  
}  
  
Properties
```

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- formNotFound - The specified form cannot be found.
- formInvalid - The specified form is invalid.

formName

Specifies the name of the form.

base

Specifies the base unit of measurement of the form as one of the following:

- inch - The base unit is inches.
- mm - The base unit is millimeters.
- rowColumn - The base unit is rows and columns.

unitX

Specifies the horizontal resolution of the base units as a fraction of the `base` value. For example, a value of 16 applied to the base unit `inch` means that the base horizontal resolution is 1/16 inch.

unitY

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

Specifies the vertical resolution of the base units as a fraction of the *base* value. For example, a value of 10 applied to the base unit *mm* means that the base vertical resolution is 0.1 mm.

width

Specifies the width of the form in terms of the base horizontal resolution.

height

Specifies the height of the form in terms of the base vertical resolution.

alignment

Specifies the relative alignment of the form on the media and can be one of the following values:

- topLeft - The form is aligned relative to the top and left edges of the media.
- topRight - The form is aligned relative to the top and right edges of the media.
- bottomLeft - The form is aligned relative to the bottom and left edges of the media.
- bottomRight - The form is aligned relative to the bottom and right edges of the media.

orientation

Specifies the orientation of the form as one of the following values:

- portrait - The orientation of the form is portrait.
- landscape - The orientation of the form is landscape.

offsetX

Specifies the horizontal offset of the position of the top-left corner of the form, relative to the left or right edge specified by *alignment*. This value is specified in terms of the base horizontal resolution and is always positive.

offsetY

Specifies the vertical offset of the position of the top-left corner of the form, relative to the top or bottom edge specified by *alignment*. This value is specified in terms of the base vertical resolution and is always positive.

versionMajor

Specifies the major version of the form. If the version is not specified in the form, then zero is returned.

versionMinor

Specifies the minor version of the form. If the version is not specified in the form, then zero is returned.

userPrompt

The user prompt string. This will be omitted if the form does not define a value for the user prompt.

fields

The field names.

Event Messages

None

[11.3.4 - Printer.GetQueryMedia](#)

This command is used to retrieve details of the definition of a specified media.

Command Message

| Payload | Type | Required |
|----------------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "mediaName": Add example to YAML | string | |

Properties

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

mediaName

The media name for which to retrieve details.

Completion Message

| <u>Payload</u> | <u>Type</u> | <u>Required</u> |
|--|-------------|-----------------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "mediaNotFound", | string | |
| "mediaType": "generic", | string | |
| "base": "inch", | string | |
| "unitX": 0, | integer | |
| "unitY": 0, | integer | |
| "sizeWidth": 0, | integer | |
| "sizeHeight": 0, | integer | |
| "pageCount": 0, | integer | |
| "lineCount": 0, | integer | |
| "printAreaX": 0, | integer | |
| "printAreaY": 0, | integer | |
| "printAreaWidth": 0, | integer | |
| "printAreaHeight": 0, | integer | |
| "restrictedAreaX": 0, | integer | |

```

"restrictedAreaY": 0,           integer
"restrictedAreaWidth": 0,       integer
"restrictedAreaHeight": 0,      integer
"stagger": 0,                  integer
"foldType": "none",            string
"paperSources": {               object
  "any": false,                boolean
  "upper": false,              boolean
  "lower": false,              boolean
  "external": false,            boolean
  "aux": false,                boolean
  "aux2": false,               boolean
  "park": false,               boolean
}
}

```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- mediaNotFound - The specified media definition cannot be found.
- mediaInvalid - The specified media definition is invalid.

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

mediaType

Specifies the type of media as one of the following:

- generic - The media is a generic media, i.e. a single sheet.
- passbook - The media is a passbook media.
- multipart - The media is a multi part media.

base

Specifies the base unit of measurement of the form and can be one of the following values:

- inch - The base unit is inches.
- mm - The base unit is millimeters.
- rowcolumn - The base unit is rows and columns.

unitX

Specifies the horizontal resolution of the base units as a fraction of the [base](#) value. For example, a value of 16 applied to the base unit *inch* means that the base horizontal resolution is 1/16th inch.

unitY

Specifies the vertical resolution of the base units as a fraction of the [base](#) value. For example, a value of 10 applied to the base unit *mm* means that the base vertical resolution is 0.1 mm.

sizeWidth

Specifies the width of the media in terms of the base horizontal resolution.

sizeHeight

Specifies the height of the media in terms of the base vertical resolution.

pageCount

Specifies the number of pages in a media of type *passbook*.

lineCount

Specifies the number of lines on a page for a media of type *passbook*.

printAreaX

Specifies the horizontal offset of the printable area relative to the top left corner of the media in terms of the base horizontal resolution.

printAreaY

Specifies the vertical offset of the printable area relative to the top left corner of the media in terms of the base vertical resolution.

printAreaWidth

Specifies the printable area width of the media in terms of the base horizontal resolution.

printAreaHeight

Specifies the printable area height of the media in terms of the base vertical resolution.

restrictedAreaX

Specifies the horizontal offset of the restricted area relative to the top left corner of the media in terms of the base horizontal resolution.

restrictedAreaY

Specifies the vertical offset of the restricted area relative to the top left corner of the media in terms of the base vertical resolution.

restrictedAreaWidth

Specifies the restricted area width of the media in terms of the base horizontal resolution.

restrictedAreaHeight

Specifies the restricted area height of the media in terms of the base vertical resolution.

stagger

Specifies the staggering from the top in terms of the base vertical resolution for a media of type *passbook*.

foldType

Specified the type of fold for a media of type *passbook* as one of the following:

- none - Passbook has no fold.
- horizontal - Passbook has a horizontal fold.
- vertical - Passbook has a vertical fold.

paperSources

Specifies the Paper sources to use when printing forms using this media as a combination of the following flags

paperSources/any

Use any paper source.

paperSources/upper

Use the only or the upper paper source.

paperSources/lower

Use the lower paper source.

paperSources/external

Use the external paper source.

paperSources/aux

Use the auxiliary paper source.

paperSources/aux2

Use the second auxiliary paper source.

`paperSources/park`

Use the parking station.

Event Messages

None

[11.3.5 - Printer.GetQueryField](#)

This command is used to retrieve details of the definition of a single or all fields on a specified form.

Command Message

| Payload | Type | Required |
|---|---------|----------|
| { | | |
| <code>"timeout": 5000,</code> | integer | |
| <code>"formName": Add example to YAML,</code> | string | |
| <code>"fieldName": Add example to YAML</code> | string | |
| } | | |

Properties

`timeout`

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

`default: 0`

`formName`

The form name.

fieldName

The name of the field about which to retrieve details. If omitted, then details are retrieved for all fields on the form.

Completion Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "formNotFound", | string | |
| "fields": { | object | |
| "indexCount": 0, | integer | |
| "type": "text", | string | |
| "class": "static", | string | |
| "access": "read", | string | |
| "overflow": "terminate", | string | |
| "initialValue": Add example to YAML, | string | |
| "format": Add example to YAML, | string | |
| "coercivity": "auto" | string | |
| } | | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- formNotFound - The specified form cannot be found.
- fieldNotFound - The specified field cannot be found.
- formInvalid - The specified form is invalid.
- fieldInvalid - The specified field is invalid.

fields

Details of the field(s) requested. For each object, the key is the field name.

fields/indexCount

Specifies the number of entries for an index field. A value of zero indicates that this field is not an index field. Index fields are typically used to present information in a tabular fashion.

fields/type

Specifies the type of field as one of the following:

- text - The field is a text field.
- micr - The field is a Magnetic Ink Character Recognition field.
- ocr - The field is an Optical Character Recognition field.
- msf - The field is a Magnetic Stripe Facility field.
- barcode - The field is a barcode field.
- graphic - The field is a Graphic field.
- pagemark - The field is a Page Mark field.

fields/class

Specifies the class of the field as one of the following:

- static - The field data cannot be set by the client.
- optional - The field data can be set by the client.
- required - The field data must be set by the client.

fields/access

Specifies the field access as one of the following:

- **read** - The field is used for input.
- **write** - The field is used for output.
- **readWrite** - The field is used for both input and output.

fields/overflow

Specifies how an overflow of field data should be handled as one of the following:

- **terminate** - Return an error and terminate printing of the form.
- **truncate** - Truncate the field data to fit in the field.
- **bestFit** - Fit the text in the field.
- **overwrite** - Print the field data beyond the extents of the field boundary.
- **wordWrap** - If the field can hold more than one line the text is wrapped around.
Wrapping is performed, where possible, by splitting the line on a space character or a hyphen character or any other character which is used to join two words together.

fields initialValue

The initial value of the field. When the form is printed (using [Printer.PrintForm](#)), this value will be used if another value is not provided. This value will be omitted if the parameter is not specified in the field definition.

fields format

Format string as defined in the form for this field. This value will be omitted if the parameter is not specified in the field definition.

fields coercivity

Specifies the coercivity to be used for writing the magnetic stripe as one of the following:

- **auto** - The coercivity is decided by the Service Provider or the hardware.
- **low** - A low coercivity is to be used for writing the magnetic stripe.
- **high** - A high coercivity is to be used for writing the magnetic stripe.

Event Messages

None

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

11.3.6 - Printer.GetCodeLineMapping

This command is used to retrieve the byte code mapping for the special banking symbols defined for image processing (e.g. check processing). This mapping must be reported as there is no standard for the fonts defined below.

Command Message

| Payload | Type | Required |
|--------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "codelineFormat": "cmc7" | string | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

codelineFormat

Specifies the code-line format that the mapping for the special characters is required for. This field can be one of the following values:

- cmc7 - Report the CMC7 mapping.
- e13b - Report the E13B mapping.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Note: work-in-progress. Use at your own risk.

```
"codelineFormat": "cmc7",           string
"charMapping": Add example to YAML   string
}
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

codelineFormat

Specifies the code-line format that is being reported as one of the following:

- cmc7 - CMC7 mapping.
 - e13b - E13B mapping.

charMapping

Defines the mapping of the font specific symbols to byte values. These byte values are used to represent the font specific characters when the code line is read through the Printer.ReadImage command. The font specific meaning of each index is defined in the following tables.

E13B

| Index | Symbol | Meaning |
|-------|--------|---------------------|
| 0 | : | Transit |
| 1 | , | Amount |
| 2 | | On Us |
| 3 | .. | Dash |
| 4 | N/A | Reject / Unreadable |

CMC7

Index Symbol Meaning

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members

| | | |
|---|-------|--------------------------------|
| 0 | ■ | S1 - Start of Bank Account |
| 1 | ■■ | S2 - Start of the Amount field |
| 2 | ■■■ | S3 - Terminate Routing |
| 3 | ■■■■ | S4 - Unused |
| 4 | ■■■■■ | S5 - Transit / Routing |
| 5 | N/A | Reject / Unreadable |

Event Messages

None

11.3.7 - Printer.ControlMedia

This command is used to control media.

If an eject operation is specified, it completes when the media is moved to the exit slot. An unsolicited event is generated when the media has been taken by the user (only if the **mediaTaken** capability is true).

Command Message

| Payload | Type | Required |
|----------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "mediaControl": { | object | |
| "eject": false, | boolean | |
| "perforate": false, | boolean | |
| "cut": false, | boolean | |
| "skip": false, | boolean | |
| "flush": false, | boolean | |
| "retract": false, | boolean | |
| "stack": false, | boolean | |
| "partialCut": false, | boolean | |

```

"alarm": false,           boolean
"forward": false,         boolean
"backward": false,        boolean
"turnMedia": false,       boolean
"stamp": false,           boolean
"park": false,            boolean
"expel": false,           boolean
"ejectToTransport": false, boolean
"rotate180": false,       boolean
"clearBuffer": false      boolean
}

}

```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

mediaControl

Specifies the manner in which the media should be handled, as a combination of the following flags:

It is not possible to combine the flags eject, retract, park, expel and ejectToTransport with each other otherwise the command completes with `errInvalidData`.

It is not possible to combine the flag `clearBuffer` with any other flags, otherwise the command completes with `invalidData`.

a client should be aware that the sequence of the actions is not guaranteed if more than one flag is specified in this parameter.

mediaControl/eject

Flush any data to the printer that has not yet been printed from previous [Printer.PrintForm](#) or [Printer.PrintRawFile](#) commands, then eject the media.

mediaControl/perforate

Flush data as per eject, then perforate the media.

mediaControl/cut

Flush data as per eject, then cut the media. For printers which have the ability to stack multiple cut sheets and deliver them as a single bundle to the customer, cut causes the media to be stacked and eject causes the bundle to be moved to the exit slot.

mediaControl/skip

Flush data as per eject, then skip the media to mark.

mediaControl/flush

Flush any data to the printer that has not yet been physically printed from previous [Printer.PrintForm](#) or [Printer.PrintRawFile](#) commands. This will synchronize the client with the device to ensure that all data has been physically printed.

mediaControl/retract

Flush data as per flush, then retract the media to retract bin number one. For devices with more than one bin the command [Printer.RetractMedia](#) should be used if the media should be retracted to another bin than bin number one.

mediaControl/stack

Flush data as per flush, then move the media item on the internal stacker.

mediaControl/partialCut

Flush the data as per flush, then partially cut the media.

mediaControl/alarm

Cause the printer to ring a bell, beep, or otherwise sound an audible alarm.

mediaControl/forward

Flush the data as per flush, then turn one page forward.

mediaControl/backward

Flush the data as per flush, then turn one page backward.

mediaControl/turnMedia

Flush the data as per flush, then turn inserted media.

mediaControl/stamp

Flush the data as per flush, then stamp on inserted media.

mediaControl/park

Park the media in the parking station.

mediaControl/expel

Flush the data as per flush, then throw the media out of the exit slot.

mediaControl/ejectToTransport

Flush the data as per flush, then move the media to a position on the transport just behind the exit slot.

mediaControl/rotate180

Flush the data as per flush, then rotate media 180 degrees in the printing plane.

mediaControl/clearBuffer

Clear any data that has not yet been physically printed from previous *PrinterPrintForm* or

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

Printer.PrintRawFile commands.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "noMediaPresent" | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- noMediaPresent - The control action could not be completed because there is no media in the device, the media is not in a position where it can be controlled, or (in the case of *retract*) has been removed.
- flushFail - The device was not able to flush data.
- retractBinFull - The retract bin is full. No more media can be retracted. The current media is still in the device.
- stackerFull - The internal stacker is full. No more media can be moved to the stacker.
- pageTurnFail - The device was not able to turn the page.
- mediaTurnFail - The device was not able to turn the inserted media.
- shutterFail - Open or close of the shutter failed due to manipulation or hardware error.
- mediaJammed - The media is jammed; operator intervention is required.

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

- paperJammed - The paper is jammed.
- paperOut - The paper supply is empty.
- inkOut - No stamping possible, stamping ink supply empty.
- tonerOut - Toner or ink supply is empty or printing contrast with ribbon is not sufficient.
- sequenceInvalid - Programming error. Invalid command sequence (e.g. *park* and the parking station is busy).
- mediaRetained - Media has been retracted in attempts to eject it. The device is clear and can be used.
- blackMark - Black mark detection has failed, nothing has been printed.
- mediaRetracted - Presented media was automatically retracted before all wads could be presented and before the command could complete successfully.

Event Messages

- [Printer.RetractBinThresholdEvent](#)
- [Printer.MediaTakenEvent](#)
- [Printer.PaperThresholdEvent](#)
- [Printer.TonerThresholdEvent](#)
- [Printer.InkThresholdEvent](#)
- [Printer.MediaPresentedEvent](#)
- [Printer.MediaAutoRetractedEvent](#)

[11.3.8 - Printer.PrintForm](#)

This command is used to print a form by merging the supplied variable field data with the defined form and field data specified in the form. If no media is present, the device waits, for the period of time specified by the [timeout](#) parameter, for media to be inserted from the external paper source.

All error codes (except [noMediaPresent](#)) and events listed under the [Printer.ControlMedia](#) command description can also occur on this command.

An invalid field name is treated as a [Printer.FieldWarningEvent](#) event with [failure notfound](#). A *Printer.FieldWarningEvent* event is returned with [fieldOverflow](#) status if the data overflows the field, and the field definition [OVERFLOW](#) value is [TRUNCATE](#), [BESTFIT](#), [OVERWRITE](#) or [WORDWRAP](#). Other field-related problems generate a field error return and event.

Command Message

| Payload | Type | Required |
|---------|------|----------|
|---------|------|----------|

```
{  
  "timeout": 5000,           integer  
  "formName": Add example to YAML, string  
  "mediaName": Add example to YAML, string  
  "alignment": "formDefinition", string  
  "offsetX": 0,              integer  
  "offsetY": 0,              integer  
  "resolution": "low",      string  
  "mediaControl": {  
    "eject": false,          boolean  
    "perforate": false,     boolean  
    "cut": false,           boolean  
    "skip": false,          boolean  
    "flush": false,         boolean  
    "retract": false,       boolean  
    "stack": false,         boolean  
    "partialCut": false,    boolean  
    "alarm": false,         boolean  
    "forward": false,       boolean  
    "backward": false,      boolean  
    "turnMedia": false,     boolean  
    "stamp": false,         boolean  
    "park": false,          boolean  
    "expel": false,         boolean  
    "ejectToTransport": false, boolean  
    "rotate180": false,     boolean
```

```
"clearBuffer": false          boolean  
},  
"fields": {                  object  
},  
"paperSource": "any"         string  
}
```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

formName

The form name.

mediaName

The media name. If no media definition applies, this should be empty or omitted.

alignment

Specifies the alignment of the form on the physical media, as one of the following values:

- `formDefinition` - Use the alignment specified in the form definition.
- `topLeft` - Align form to top left of physical media.
- `topRight` - Align form to top right of physical media.
- `bottomLeft` - Align form to bottom left of physical media.
- `bottomRight` - Align form to bottom right of physical media.

offsetX

Specifies the horizontal offset of the form, relative to the horizontal alignment specified in `alignment`, in horizontal resolution units (from form definition); always a positive number

(i.e. if aligned to the right side of the media, means offset the form to the left). A value of *formDefinition* indicates that the *xoffset* value from the form definition should be used.

offsetY

Specifies the vertical offset of the form, relative to the vertical alignment specified in *alignment*, in vertical resolution units (from form definition); always a positive number (i.e. if aligned to the bottom of the media, means offset the form upward). A value of *formDefinition* indicates that the *yoffset* value from the form definition should be used.

resolution

Specifies the resolution in which to print the form. Possible values are:

- low - Print form with low resolution.
- medium - Print form with medium resolution.
- high - Print form with high resolution.
- veryHigh - Print form with very high resolution.

mediaControl

Specifies the manner in which the media should be handled after the printing is done, as a combination of the following flags. If no flags are set, it means do none of these actions, as when printing multiple forms on a single page. When no flags are set and the device does not support the flush capability, the data will be printed immediately. If the device supports flush, the data may be buffered and the [Printer.ControlMedia](#) command should be used to synchronize the client with the device to ensure that all data has been physically printed. The [clearBuffer](#) flag is not applicable to this command. If set, the command will fail with error *invalidData*.

mediaControl/eject

Flush any data to the printer that has not yet been printed from previous [Printer.PrintForm](#) or [Printer.PrintRawFile](#) commands, then eject the media.

mediaControl/perforate

Flush data as per eject, then perforate the media.

mediaControl/cut

Flush data as per eject, then cut the media. For printers which have the ability to stack

multiple cut sheets and deliver them as a single bundle to the customer, cut causes the media to be stacked and eject causes the bundle to be moved to the exit slot.

mediaControl/skip

Flush data as per eject, then skip the media to mark.

mediaControl/flush

Flush any data to the printer that has not yet been physically printed from previous *Printer.PrintForm* or *Printer.PrintRawFile* commands. This will synchronize the client with the device to ensure that all data has been physically printed.

mediaControl/retract

Flush data as per flush, then retract the media to retract bin number one. For devices with more than one bin the command [Printer.RetractMedia](#) should be used if the media should be retracted to another bin than bin number one.

mediaControl/stack

Flush data as per flush, then move the media item on the internal stacker.

mediaControl/partialCut

Flush the data as per flush, then partially cut the media.

mediaControl/alarm

Cause the printer to ring a bell, beep, or otherwise sound an audible alarm.

mediaControl/forward

Flush the data as per flush, then turn one page forward.

mediaControl/backward

Flush the data as per flush, then turn one page backward.

mediaControl/turnMedia

Flush the data as per flush, then turn inserted media.

mediaControl/stamp

Flush the data as per flush, then stamp on inserted media.

mediaControl/park

Park the media in the parking station.

mediaControl/expel

Flush the data as per flush, then throw the media out of the exit slot.

mediaControl/ejectToTransport

Flush the data as per flush, then move the media to a position on the transport just behind the exit slot.

mediaControl/rotate180

Flush the data as per flush, then rotate media 180 degrees in the printing plane.

mediaControl/clearBuffer

Clear any data that has not yet been physically printed from previous *Printer.PrintForm* or *Printer.PrintRawFile* commands.

fields

An object containing one or more key/value pairs where the key is a field name and the value is the field value. If the field is an index field, the key must be specified as *fieldname/index* where index specifies the zero-based element of the index field. The field names and values can contain UNICODE if supported by the service.

paperSource

Specifies the Paper source to use when printing this form. When the value is zero, then the

paper source is determined from the media definition. This parameter is ignored if there is already paper in the print position. Possible values are:

- any - Any paper source can be used; it is determined by the service.
- upper - Use the only paper source or the upper paper source, if there is more than one paper supply.
- lower - Use the lower paper source.
- external - Use the external paper source (such as envelope tray or single sheet feed).
- aux - Use the auxiliary paper source.
- aux2 - Use the second auxiliary paper source.
- park - Use the parking station.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "formNotFound" | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- formNotFound - The specified form definition cannot be found.
- flushFail - The device was not able to flush data.

- mediaOverflow - The form overflowed the media.
- fieldSpecFailure - The syntax of the [fields](#) member is invalid.
- fieldError - An error occurred while processing a field, causing termination of the print request. A [Printer.FieldErrorEvent](#) event is posted with the details.
- mediaNotFound - The specified media definition cannot be found.
- mediaInvalid - The specified media definition is invalid.
- formInvalid - The specified form definition is invalid.
- mediaSkewed - The media skew exceeded the limit in the form definition.
- retractBinFull - The retract bin is full. No more media can be retracted. The current media is still in the device.
- stackerFull - The internal stacker is full. No more media can be moved to the stacker.
- pageTurnFail - The device was not able to turn the page.
- mediaTurnFail - The device was not able to turn the inserted media.
- shutterFail - Open or close of the shutter failed due to manipulation or hardware error.
- mediaJammed - The media is jammed; operator intervention is required.
- charSetData - Character set(s) supported by Service Provider is inconsistent with use of *fields*.
- paperJammed - The paper is jammed.
- paperOut - The paper supply is empty.
- inkOut - No stamping possible, stamping ink supply empty.
- tonerOut - Toner or ink supply is empty or printing contrast with ribbon is not sufficient.
- sequenceInvalid - Programming error. Invalid command sequence (e.g. [mediaControl](#) = park and park position is busy).
- sourceInvalid - The selected paper source is not supported by the hardware.
- mediaRetained - Media has been retracted in attempts to eject it. The device is clear and can be used.
- blackMark - Black mark detection has failed, nothing has been printed.
- mediaSize - The media entered has an incorrect size and the media remains inside the device.
- mediaRejected - The media was rejected during the insertion phase and no data has been printed. The [Printer.MediaRejectedEvent](#) event is posted with the details. The device is still operational.
- mediaRetracted - Presented media was automatically retracted before all wads could be presented and before the command could complete successfully.
- msfError - An error occurred while writing the magnetic stripe data.
- noMSF - No magnetic stripe found; media may have been inserted or pulled through the wrong way.

Event Messages

- [Printer.NoMediaEvent](#)
- [Printer.MediaInsertedEvent](#)
- [Printer.FieldErrorEvent](#)
- [Printer.FieldWarningEvent](#)
- [Printer.RetractBinThresholdEvent](#)
- [Printer.MediaTakenEvent](#)
- [Printer.PaperThresholdEvent](#)
- [Printer.TonerThresholdEvent](#)
- [Printer.InkThresholdEvent](#)
- [Printer.MediaPresentedEvent](#)
- [Printer.MediaRejectedEvent](#)
- [Printer.MediaAutoRetractedEvent](#)

[11.3.9 - Printer.ReadForm](#)

This command is used to read data from input fields on the specified form. These input fields may consist of MICR, OCR, MSF, BARCODE, or PAGEMARK input fields. These input fields may also consist of TEXT fields for purposes of detecting available passbook print lines with passbook printers supporting such capability. If no media is present, the device waits, for the [timeout](#) specified, for media to be inserted.

All error codes (except *noMediaPresent*) and events listed under the [Printer.ControlMedia](#) command description can also occur on this command.

The following applies to the usage of *fields* for passbook: If the media type is PASSBOOK, and the field(s) type is TEXT, and the Service Provider and the underlying passbook printer are capable of detecting available passbook print lines, then the field(s) will be returned without a value, in the format "" or *fieldname/index*, if the field is available for passbook printing. Field(s) unavailable for passbook printing will not be returned. The Service Provider will examine the passbook text field(s) supplied in the *fieldNames* field, and with the form/*fields* definition and the underlying passbook printer capability determine which fields should be available for passbook printing.

To illustrate when media type is PASSBOOK, if a form named PSBKTST1 contains 24 fields, one field per line, and the field names are LINE1 through LINE24 (same order as printing), and after execution of this command *fields* contains fields LINE13 through LINE24, then the first print line available for passbook printing is line 13.

To illustrate another example when media type is PASSBOOK, if a form named PSBKTST2 contains 24 fields, one field per line, and the field names are LINE1 through LINE24 (same order as printing), and after execution of this command *fields* contains fields LINE13, and LINE20 through LINE24 then the first print line available for passbook printing is line 13, however lines 14-19 are not also available, so if the client is attempting to determine the

first available print line after which all subsequent print lines are also available then line 20 is a better choice.

Command Message

| Payload | Type | Required |
|--------------------------------------|----------------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "formName": Add example to YAML, | string | |
| "fieldNames": [Add example to YAML], | array (string) | |
| "mediaName": Add example to YAML, | string | |
| "mediaControl": { | object | |
| "eject": false, | boolean | |
| "perforate": false, | boolean | |
| "cut": false, | boolean | |
| "skip": false, | boolean | |
| "flush": false, | boolean | |
| "retract": false, | boolean | |
| "stack": false, | boolean | |
| "partialCut": false, | boolean | |
| "alarm": false, | boolean | |
| "forward": false, | boolean | |
| "backward": false, | boolean | |
| "turnMedia": false, | boolean | |
| "stamp": false, | boolean | |
| "park": false, | boolean | |
| "expel": false, | boolean | |
| "ejectToTransport": false, | boolean | |
| "rotate180": false, | boolean | |

```
"clearBuffer": false          boolean  
}  
}
```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

formName

The name of the form.

fieldNames

The field names from which to read input data. If this is omitted or empty, all input fields on the form will be read.

mediaName

The media name. If omitted or empty, no media definition applies.

mediaControl

Specifies the manner in which the media should be handled after the reading was done and can be a combination of the following flags. The [clearBuffer](#) flag is not applicable to this command.

mediaControl/eject

Flush any data to the printer that has not yet been printed from previous [Printer.PrintForm](#) or [Printer.PrintRawFile](#) commands, then eject the media.

mediaControl/perforate

Flush data as per eject, then perforate the media.

mediaControl/cut

Flush data as per eject, then cut the media. For printers which have the ability to stack multiple cut sheets and deliver them as a single bundle to the customer, cut causes the media to be stacked and eject causes the bundle to be moved to the exit slot.

mediaControl/skip

Flush data as per eject, then skip the media to mark.

mediaControl/flush

Flush any data to the printer that has not yet been physically printed from previous *Printer.PrintForm* or *Printer.PrintRawFile* commands. This will synchronize the client with the device to ensure that all data has been physically printed.

mediaControl/retract

Flush data as per flush, then retract the media to retract bin number one. For devices with more than one bin the command [Printer.RetractMedia](#) should be used if the media should be retracted to another bin than bin number one.

mediaControl/stack

Flush data as per flush, then move the media item on the internal stacker.

mediaControl/partialCut

Flush the data as per flush, then partially cut the media.

mediaControl/alarm

Cause the printer to ring a bell, beep, or otherwise sound an audible alarm.

mediaControl/forward

Flush the data as per flush, then turn one page forward.

mediaControl/backward

Flush the data as per flush, then turn one page backward.

mediaControl/turnMedia

Flush the data as per flush, then turn inserted media.

mediaControl/stamp

Flush the data as per flush, then stamp on inserted media.

mediaControl/park

Park the media in the parking station.

mediaControl/expel

Flush the data as per flush, then throw the media out of the exit slot.

mediaControl/ejectToTransport

Flush the data as per flush, then move the media to a position on the transport just behind the exit slot.

mediaControl/rotate180

Flush the data as per flush, then rotate media 180 degrees in the printing plane.

mediaControl/clearBuffer

Clear any data that has not yet been physically printed from previous *Printer.PrintForm* or *Printer.PrintRawFile* commands.

Completion Message

| <u>Payload</u> | <u>Type</u> | <u>Required</u> |
|----------------|-------------|-----------------|
| { | | |

```
"completionCode": "success",           string
"errorDescription": Add example to YAML,  string
"errorCode": "formNotFound",           string
"fields": {                           object
}
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- formNotFound - The specified form cannot be found.
- readNotSupported - The device has no read capability.
- fieldSpecFailure - The syntax of the `fieldNames` member is invalid.
- fieldError - An error occurred while processing a field, causing termination of the print request. A `Printer.FieldErrorEvent` event is posted with the details.
- mediaNotFound - The specified media definition cannot be found.
- mediaInvalid - The specified media definition is invalid.
- formInvalid - The specified form definition is invalid.
- mediaSkewed - The media skew exceeded the limit in the form definition.
- retractBinFull - The retract bin is full. No more media can be retracted. The current media is still in the device.
- shutterFail - Open or close of the shutter failed due to manipulation or hardware error.
- mediaJammed - The media is jammed.
- inkOut - No stamping possible, stamping ink supply empty.
- lampInoperative - Imaging lamp is inoperative.

- sequenceInvalid - Programming error. Invalid command sequence (e.g. [mediaControl](#) = park and park position is busy).
- mediaSize - The media entered has an incorrect size.
- mediaRejected - The media was rejected during the insertion phase. The [Printer.MediaRejectedEvent](#) event is posted with the details. The device is still operational.
- msfError - The MSF read operation specified by the forms definition could not be completed successfully due to invalid magnetic stripe data.
- noMSF - No magnetic stripe found; media may have been inserted or pulled through the wrong way.

fields

An object containing one or more key/value pairs where the key is a field name and the value is the field value. If the field is an index field, the key must be specified as *fieldname/index* where index specifies the zero-based element of the index field. The field names and values can contain UNICODE if supported by the service.

Event Messages

- [Printer.NoMediaEvent](#)
- [Printer.MediaInsertedEvent](#)
- [Printer.FieldErrorEvent](#)
- [Printer.FieldWarningEvent](#)
- [Printer.RetractBinThresholdEvent](#)
- [Printer.MediaTakenEvent](#)
- [Printer.InkThresholdEvent](#)
- [Printer.LampThresholdEvent](#)
- [Printer.MediaRejectedEvent](#)

[11.3.10 - Printer.RawData](#)

This command is used to send raw data (a byte string of device dependent data) to the physical device.

Clients which send raw data to a device will typically not be device or vendor independent. Problems with the use of this command include:

1. The data sent to the device can include commands that change the state of the device in unpredictable ways (in particular, in ways that the Service Provider may not be aware of).
2. Usage of this command will not be portable.

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

3. This command violates the XFS forms model that is the basis of XFS printer access.
Thus usage of this command should be avoided whenever possible.

Command Message

| Payload | Type | Required |
|------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "inputData": "no", | string | |
| "data": "UmF3RGF0YQ==" | string | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

inputData

Specifies that input data from the device is expected in response to sending the raw data (i.e. the data contains a command requesting data). Possible values are:

- no - No input data is expected.
- yes - Input data is expected.

data

BASE64 encoded device dependent data to be sent to the device.

Completion Message

| Payload | Type | Required |
|---------|------|----------|
| { | | |

```
"completionCode": "success",           string  
"errorDescription": Add example to YAML,  string  
"errorCode": "shutterFail",            string  
"data": "UmF3RGF0YQ=="                string  
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- shutterFail - Open or close of the shutter failed due to manipulation or hardware error.
- mediaJammed - The media is jammed.
- paperJammed - The paper is jammed.
- paperOut - The paper supply is empty.
- tonerOut - Toner or ink supply is empty or printing contrast with ribbon is not sufficient.
- mediaRetained - Media has been retracted in attempts to eject it. The device is clear and can be used.
- blackMark - Black mark detection has failed, nothing has been printed.
- mediaRetracted - Presented media was automatically retracted before all wads could be presented and before the command could complete successfully.

data

BASE64 encoded device dependent data received from the device.

Event Messages

- [Printer.RetractBinThresholdEvent](#)
- [Printer.MediaTakenEvent](#)
- [Printer.PaperThresholdEvent](#)
- [Printer.TonerThresholdEvent](#)
- [Printer.MediaPresentedEvent](#)
- [Printer.MediaAutoRetractedEvent](#)

11.3.11 - Printer.MediaExtents

This command is used to get the extents of the media inserted in the physical device. The input parameter specifies the base unit and fractions in which the media extent values will be returned. If no media is present, the device waits, for the [timeout](#) specified, for media to be inserted.

Command Message

| <u>Payload</u> | <u>Type</u> | <u>Required</u> |
|---|-------------|-----------------|
| { "timeout": 5000, integer "base": "inches", string "unitX": 0, integer "unitY": 0 integer } | | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

base

Specifies the base unit of measurement of the media and can be one of the following values:

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

- inches - The base unit is inches.
- mm - The base unit is millimeters.
- rowColumn - The base unit is rows and columns.

unitX

Specifies the horizontal resolution of the base units as a fraction of the base value. For example, a value of 16 applied to the base unit, inches, means that the base horizontal resolution is 1/16.

unitY

Specifies the vertical resolution of the base units as a fraction of the base value. For example, a value of 10 applied to the base unit, mm, means that the base vertical resolution is 0.1 mm.

Completion Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "extentNotSupported", | string | |
| "sizeX": 0, | integer | |
| "sizeY": 0 | integer | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional

information

errorCode

Specifies the error code if applicable. The following values are possible:

- extentNotSupported - The device cannot report extent(s).
- mediaJammed - The media is jammed.
- lampInoperative - Imaging lamp is inoperative.
- mediaSize - The media entered has an incorrect size and the media remains inside the device.
- mediaRejected - The media was rejected during the insertion phase. The [Printer.MediaRejectedEvent](#) event is posted with the details. The device is still operational.

sizeX

Specifies the width of the media in terms of the base horizontal resolution.

sizeY

Specifies the height of the media in terms of the base vertical resolution.

Event Messages

- [Printer.NoMediaEvent](#)
- [Printer.MediaInsertedEvent](#)
- [Printer.MediaRejectedEvent](#)
- [Printer.MediaTakenEvent](#)

[11.3.12 - Printer.ResetCount](#)

This function resets the present value for number of media items retracted to zero. The function is possible only for printers with retract capability.

The number of media items retracted is controlled by the service and can be requested before resetting using the info [Common.Status](#) command.

Command Message

| Payload | Type | Required |
|---------|------|----------|
|---------|------|----------|

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

```
{
  "timeout": 5000,  integer
  "binNumber": 0   integer
}
```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

binNumber

Specifies the height of the media in terms of the base vertical resolution.

Completion Message

| Payload | Type | Required |
|---|--------|----------|
| <pre>{ "completionCode": "success", "errorDescription": Add example to YAML }</pre> | string | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional

information

Event Messages

- [Printer.RetractBinThresholdEvent](#)

[11.3.13 - Printer.ReadImage](#)

This function returns image data from the current media. If no media is present, the device waits for the timeout specified, for media to be inserted.

If the returned image data is in Windows bitmap format (BMP) and a file path for storing the image is not supplied, then the first byte of data will be the start of the Bitmap Info Header (this bitmap format is known as DIB, Device Independent Bitmap). The Bitmap File Info Header, which is only present in file versions of bitmaps, will NOT be returned. If the returned image data is in bitmap format (BMP) and a file path for storing the image is supplied, then the first byte of data in the stored file will be the Bitmap File Info Header.

Command Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "frontImageType": "tif", | string | |
| "backImageType": "tif", | string | |
| "frontImageColorFormat": "binary", | string | |
| "backImageColorFormat": "binary", | string | |
| "codelineFormat": "cmc7", | string | |
| "imageSource": { | object | |
| "front": false, | boolean | |
| "back": false, | boolean | |
| "codeline": false | boolean | |
| }, | | |
| "frontImageFile": Add example to YAML, string | | |

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

"backImageFile": Add example to YAML string

}

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

frontImageType

Specifies the format of the front image returned by this command as one of the following. If omitted, no front image is returned.

- tif - The returned image is in TIF 6.0 format.
- wmf - The returned image is in WMF (Windows Metafile) format.
- bmp - The returned image is in BMP format.
- jpg - The returned image is in JPG format.

backImageType

Specifies the format of the back image returned by this command as one of the following. If omitted, no back image is returned.

- tif - The returned image is in TIF 6.0 format.
- wmf - The returned image is in WMF (Windows Metafile) format.
- bmp - The returned image is in BMP format.
- jpg - The returned image is in JPG format.

frontImageColorFormat

Specifies the color format of the requested front image as one of the following:

- binary - The scanned images has to be returned in binary (image contains two colors, usually the colors black and white).
- grayscale - The scanned images has to be returned in gray scale (image contains multiple gray colors).
- fullcolor - The scanned images has to be returned in full color (image contains colors like red, green, blueetc.).

backImageColorFormat

Specifies the color format of the requested back image as one of the following:

- binary - The scanned images has to be returned in binary (image contains two colors, usually the colors black and white).
- grayscale - The scanned images has to be returned in gray scale (image contains multiple gray colors).
- fullcolor - The scanned images has to be returned in full color (image contains colors like red, green, blue etc.).

codelineFormat

Specifies the code line (MICR data) format, as one of the following flags (zero if source not selected):

- cmc7 - Read CMC7 code line.
- e13b - Read E13B code line.
- ocr - Read code line using OCR.

imageSource

Specifies the source as a combination of the following flags:

imageSource/front

The front image of the document is requested.

imageSource/back

The back image of the document is requested.

imageSource/codeline

The code line of the document is requested.

frontImageFile

File specifying where to store the front image, e.g. "C:\Temp\FrontImage.bmp". If omitted or empty, the front image data will be returned in the output parameter. This value must not contain UNICODE characters.

To reduce the size of data sent between the client and service, it is recommended to use of this parameter.

backImageFile

File specifying where to store the back image, e.g. "C:\Temp\BackImage.bmp". If omitted or empty, the back image data will be returned in the output parameter. This value must not contain UNICODE characters.

To reduce the size of data sent between the client and service, it is recommended to use of this parameter.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "shutterFail", | string | |
| "images": { | object | |
| "front": { | object | |
| "status": "ok", | string | |
| "data": Add example to YAML | string | |
| }, | | |
| "back": { | object | |
| See front properties. | | |
| }, | | |
| "codeline": { | object | |
| See front properties. | | |
| } | | |
| } | | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- shutterFail - Open or close of the shutter failed due to manipulation or hardware error.
- mediaJammed - The media is jammed; operator intervention is required.
- fileIOError - Directory does not exist or a File IO error occurred while storing the image to the hard disk.
- lampInoperative - Imaging lamp is inoperative.
- mediaSize - The media entered has an incorrect size and the media remains inside the device.
- mediaRejected - The media was rejected during the insertion phase. The [Printer.MediaRejectedEvent](#) event is posted with the details. The device is still operational.

images

The status and data for each of the requested images.

images/front

The front image status and data.

images/front/status

Status of data source. Possible values are:

- ok - The data is OK.
- notSupp - The data source is not supported.

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

- missing - The data source is missing, for example, the Service is unable to get the code line.

images/front/data

If the image source is front or back and the image data has not been stored to the hard disk (file name not provided), this will contain the Base64 encoded image.

If the image source is codeline, this contains characters in the ASCII range. If the code line was read using the OCR-A font then the ASCII codes will conform to Figure E1 in ANSI X3.17-1981. If the code line was read using the OCR-B font then the ASCII codes will conform to Figure C2 in ANSI X3.49-1975. In both these cases unrecognized characters will be reported as the REJECT code, 0x1A. The E13B and CMC7 fonts use the ASCII equivalents for the standard characters and use the byte values as reported by the Printer.CodelineMapping command for the symbols that are unique to MICR fonts.

images/back

The back image status and data.

images/codeline

The codeline status and data.

Event Messages

- [Printer.NoMediaEvent](#)
- [Printer.MediaInsertedEvent](#)
- [Printer.MediaTakenEvent](#)
- [Printer.LampThresholdEvent](#)
- [Printer.MediaRejectedEvent](#)
- [Printer.MediaAutoRetractedEvent](#)

[11.3.14 - Printer.Reset](#)

This command is used by the client to perform a hardware reset which will attempt to return the device to a known good state.

The device will attempt to retract or eject any items found anywhere within the device. This may not always be possible because of hardware problems. The [Printer.MediaDetectedEvent](#) event will inform the client where items were actually moved to.

Command Message

| Payload | Type | Required |
|--------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "mediaControl": "eject", | string | |
| "retractBinNumber": 0 | integer | |
| } | | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

mediaControl

Specifies the manner in which the media should be handled, as one of the following:

- eject - Eject the media.
- retract - Retract the media to retract bin number specified.
- expel - Throw the media out of the exit slot.

retractBinNumber

Number of the retract bin the media is retracted to. This number has to be between one and the [number of bins](#) supported by this device. It is only relevant if [mediaControl](#) is *retract*.

Completion Message

| Payload | Type | Required |
|------------------------------|--------|----------|
| { | | |
| "completionCode": "success", | string | |

```
"errorDescription": Add example to YAML,  string  
"errorCode": "shutterFail"          string  
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- shutterFail - Open or close of the shutter failed due to manipulation or hardware error.
- retractBinFull - The retract bin is full; no more media can be retracted. The current media is still in the device.
- mediaJammed - The media is jammed; operator intervention is required.
- paperJammed - The paper is jammed.

Event Messages

- [Printer.MediaDetectedEvent](#)
- [Printer.RetractBinThresholdEvent](#)
- [Printer.MediaAutoRetractedEvent](#)
- [Printer.MediaPresentedEvent](#)

[11.3.15 - Printer.RetractMedia](#)

The media is removed from its present position (media inserted into device, media entering, unknown position) and stored in one of the retract bins. An event is sent if the storage capacity of the specified retract bin is reached. If the bin is already full and the

command cannot be executed, an error is returned and the media remains in its present position.

If a retract request is received for a device with no retract capability, the unsupportedCommand error is returned.

Command Message

| Payload | Type | Required |
|------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "binNumber": 0 | integer | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

binNumber

This number has to be between one and the number of bins supported by this device. If omitted, the media will be retracted to the transport. After it has been retracted to the transport, in a subsequent operation the media can be ejected again, or retracted to one of the retract bins.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "noMediaPresent", | string | |

```
"binNumber": 0          integer  
}  
  
Properties
```

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- noMediaPresent - No media present on retract. Either there was no media present (in a position to be retracted from) when the command was called or the media was removed during the retract.
- retractBinFull - The retract bin is full; no more media can be retracted. The current media is still in the device.
- mediaJammed - The media is jammed; operator intervention is required.

binNumber

The number of the retract bin where the media has actually been deposited.

Event Messages

- Printer.RetractBinThresholdEvent

[11.3.16 - Printer.DispensePaper](#)

This command is used to move paper (which can also be a new passbook) from a paper source into the print position.

Command Message

| Payload | Type | Required |
|----------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "paperSource": "any" | string | |
| } | | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

paperSource

The paper source to dispense from. It can be one of the following:

- any - Any paper source can be used; it is determined by the service.
- upper - Use the only paper source or the upper paper source, if there is more than one paper supply.
- lower - Use the lower paper source.
- internal - Use the external paper.
- aux - Use the auxiliary paper source.
- aux2 - Use the second auxiliary paper source.
- park - Use the parking station paper source.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "paperJammed" | string | |

}

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- paperJammed - The paper is jammed.
- paperOut - The paper supply is empty.
- sequenceInvalid - Programming error. Invalid command sequence (e.g. there is already media in the print position).
- sourceInvalid - The selected paper source is not supported by the hardware.
- mediaRetracted - Presented media was automatically retracted before all wads could be presented and before the command could complete successfully.

Event Messages

- [Printer.PaperThresholdEvent](#)
- [Printer.MediaPresentedEvent](#)
- [Printer.MediaAutoRetractedEvent](#)

[11.3.17 - Printer.PrintRawFile](#)

This command is used to print a file that contains a complete print job in the native printer language. The creation and content of this file are both Operating System and printer specific and outwith the scope of this specification.

If no media is present, the device waits, for the [timeout](#) specified, for media to be inserted from the external paper source.

This command must not complete until all pages have been presented to the customer.

Printing of multiple pages is handled as described in [Command and Event Flows during Single and Multi-Page / Wad Printing](#).

Command Message

| Payload | Type | Required |
|----------------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "fileName": Add example to YAML, | string | |
| "mediaControl": { | object | |
| "eject": false, | boolean | |
| "perforate": false, | boolean | |
| "cut": false, | boolean | |
| "skip": false, | boolean | |
| "flush": false, | boolean | |
| "retract": false, | boolean | |
| "stack": false, | boolean | |
| "partialCut": false, | boolean | |
| "alarm": false, | boolean | |
| "forward": false, | boolean | |
| "backward": false, | boolean | |
| "turnMedia": false, | boolean | |
| "stamp": false, | boolean | |
| "park": false, | boolean | |
| "expel": false, | boolean | |
| "ejectToTransport": false, | boolean | |
| "rotate180": false, | boolean | |
| "clearBuffer": false | boolean | |
| } | | |

```
"paperSource": "any"          string  
}  
  
Properties
```

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

fileName

This is the full path and file name of the file to be printed. This value cannot contain UNICODE characters.

mediaControl

Specifies the manner in which the media should be handled after each page is printed, as a combination of the following flags. If no flags are set, no actions will be performed, as when printing multiple pages on a single media item. Note that the [clearBuffer](#) flag is not applicable to this command and will be ignored.

mediaControl/eject

Flush any data to the printer that has not yet been printed from previous [Printer.PrintForm](#) or [Printer.PrintRawFile](#) commands, then eject the media.

mediaControl/perforate

Flush data as per eject, then perforate the media.

mediaControl/cut

Flush data as per eject, then cut the media. For printers which have the ability to stack multiple cut sheets and deliver them as a single bundle to the customer, cut causes the media to be stacked and eject causes the bundle to be moved to the exit slot.

mediaControl/skip

Flush data as per eject, then skip the media to mark.

mediaControl/flush

Flush any data to the printer that has not yet been physically printed from previous *Printer.PrintForm* or *Printer.PrintRawFile* commands. This will synchronize the client with the device to ensure that all data has been physically printed.

mediaControl/retract

Flush data as per flush, then retract the media to retract bin number one. For devices with more than one bin the command [Printer.RetractMedia](#) should be used if the media should be retracted to another bin than bin number one.

mediaControl/stack

Flush data as per flush, then move the media item on the internal stacker.

mediaControl/partialCut

Flush the data as per flush, then partially cut the media.

mediaControl/alarm

Cause the printer to ring a bell, beep, or otherwise sound an audible alarm.

mediaControl/forward

Flush the data as per flush, then turn one page forward.

mediaControl/backward

Flush the data as per flush, then turn one page backward.

mediaControl/turnMedia

Flush the data as per flush, then turn inserted media.

mediaControl/stamp

Flush the data as per flush, then stamp on inserted media.

mediaControl/park

Park the media in the parking station.

mediaControl/expel

Flush the data as per flush, then throw the media out of the exit slot.

mediaControl/ejectToTransport

Flush the data as per flush, then move the media to a position on the transport just behind the exit slot.

mediaControl/rotate180

Flush the data as per flush, then rotate media 180 degrees in the printing plane.

mediaControl/clearBuffer

Clear any data that has not yet been physically printed from previous *Printer.PrintForm* or *Printer.PrintRawFile* commands.

paperSource

Specifies the paper source to use when printing. If omitted, the Service Provider will determine the paper source that will be used. This parameter is ignored if there is already paper in the print position. Possible values are:

- any - Any paper source can be used; it is determined by the service.
- upper - Use the only paper source or the upper paper source, if there is more than one paper supply.
- lower - Use the lower paper source.
- external - Use the external paper source (such as envelope tray or single sheet feed).
- aux - Use the auxiliary paper source.
- aux2 - Use the second auxiliary paper source.
- park - Use the parking station.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "fileNotFound" | string | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- fileNotFound - The specified file cannot be found.
- shutterFail - Open or close of the shutter failed due to manipulation or hardware error.
- mediaJammed - The media is jammed; operator intervention is required.
- paperJammed - The paper is jammed.
- paperOut - The paper supply is empty.
- tonerOut - Toner or ink supply is empty or printing contrast with ribbon is not sufficient.
- fileIOError - Directory does not exist or a File IO error occurred while processing the file.
- noMediaPresent - No media is present in the device.
- flushFail - The device was not able to flush data.
- retractBinFull - The retract bin is full. No more media can be retracted. The current media is still in the device.

- stackerFull - The internal stacker is full. No more media can be moved to the stacker.
- pageTurnFail - The device was not able to turn the page.
- mediaTurnFail - The device was not able to turn the inserted media.
- inkOut - No stamping possible, stamping ink supply empty.
- sequenceInvalid - Programming error. Invalid command sequence (e.g. *park* and the parking station is busy).
- mediaOverflow - The print request has overflowed the print media (e.g. print on a single sheet printer exceeded one page).
- mediaRetained - Media has been retracted in attempts to eject it. The device is clear and can be used.
- blackMark - Black mark detection has failed, nothing has been printed.
- sourceInvalid - The selected paper source is not supported by the hardware.
- mediaRejected - The media was rejected during the insertion phase and no data has been printed. The [Printer.MediaRejectedEvent](#) event is posted with the details. The device is still operational.
- mediaRetracted - Presented media was automatically retracted before all wads could be presented and before the command could complete successfully.

Event Messages

- [Printer.NoMediaEvent](#)
- [Printer.MediaInsertedEvent](#)
- [Printer.MediaPresentedEvent](#)
- [Printer.MediaTakenEvent](#)
- [Printer.PaperThresholdEvent](#)
- [Printer.TonerThresholdEvent](#)
- [Printer.RetractBinThresholdEvent](#)
- [Printer.InkThresholdEvent](#)
- [Printer.MediaRejectedEvent](#)
- [Printer.MediaAutoRetractedEvent](#)

[11.3.18 - Printer.LoadDefinition](#)

This command is used to load a form (including sub-forms and frames) or media definition into the list of available forms. Once a form or media definition has been loaded through this command it can be used by any of the other form/media definition processing commands. Forms and media definitions loaded through this command are persistent. When a form or media definition is loaded a [Printer.DefinitionLoadedEvent](#) event is generated to inform clients that a form or media definition has been added or replaced.

Command Message

| Payload | Type | Required |
|----------------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "fileName": Add example to YAML, | string | |
| "overwrite": false | boolean | |
| } | | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

fileName

This is the full path and file name of the file to be loaded. This value cannot contain UNICODE characters. The file contains the form (including sub-forms and frames) or media definition in text format as described in [Form, Sub-Form, Field, Frame, Table and Media Definitions](#). Only one form or media definition can be defined in the file.

overwrite

Specifies if an existing form or media definition with the same name is to be replaced. If this flag is true then an existing form or media definition with the same name will be replaced, unless the command fails with an error, where the definition will remain unchanged. If this flag is false this command will fail with an error if the form or media definition already exists.

Completion Message

| Payload | Type | Required |
|------------------------------|--------|----------|
| { | | |
| "completionCode": "success", | string | |

```
"errorDescription": Add example to YAML,  string  
"errorCode": "fileNotFound"          string  
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- fileNotFound - The specified file cannot be found.
- formInvalid - The form is invalid.
- mediaInvalid - The media definition is invalid.
- definitionExists - The specified form or media definition already exists and the *overwrite* flag was false.

Event Messages

- [Printer.DefinitionLoadedEvent](#)

[11.3.19 - Printer.SupplyReplenish](#)

After the supplies have been replenished, this command is used to indicate that one or more supplies have been replenished and are expected to be in a healthy state.

Hardware that cannot detect the level of a supply and reports on the supply's status using metrics (or some other means), must assume the supply has been fully replenished after this command is issued. The appropriate threshold event must be broadcast.

Hardware that can detect the level of a supply must update its status based on its sensors, generate a threshold event if appropriate, and succeed the command even if the supply has

not been replenished. If it has already detected the level and reported the threshold before this command was issued, the command must succeed and no threshold event is required.

If any one of the specified supplies is not supported by a Service Provider, unsupportedData should be returned, and no replenishment actions will be taken by the Service Provider.

Command Message

| Payload | Type | Required |
|------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "upper": false, | boolean | |
| "lower": false, | boolean | |
| "aux": false, | boolean | |
| "aux2": false, | boolean | |
| "toner": false, | boolean | |
| "ink": false, | boolean | |
| "lamp": false | boolean | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

upper

The only paper supply or the upper paper supply was replenished.

lower

The lower paper supply was replenished.

aux

The auxiliary paper supply was replenished.

aux2

The second auxiliary paper supply was replenished.

toner

The toner supply was replenished.

ink

The ink supply was replenished.

lamp

The imaging lamp was replaced.

Completion Message

| Payload | Type | Required |
|---|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

Event Messages

- [Printer.PaperThresholdEvent](#)
- [Printer.TonerThresholdEvent](#)
- [Printer.InkThresholdEvent](#)
- [Printer.LampThresholdEvent](#)

11.3.20 - Printer.ControlPassbook

This command can turn the pages of a passbook inserted in the printer by a specified number of pages in a specified direction and it can close the passbook. The [controlPassbook](#) field returned by [Common.Capabilities](#) specifies which functionality is supported. This command flushes the data before the pages are turned or the passbook is closed.

Command Message

| Payload | Type | Required |
|----------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "action": "forward", | string | |
| "count": 0 | integer | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

action

Specifies the direction of the page turn as one of the following values:

- forward - Turns forward the pages of the passbook.
- backward - Turns backward the pages of the passbook.
- closeForward - Close the passbook forward.
- closeBackward - Close the passbook backward.

count

Specifies the number of pages to be turned. In the case where **action** is *closeForward* or *closeBackward**, this field will be ignored.

Completion Message

| Payload | Type | Required |
|--|-------------|-----------------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "noMediaPresent" | string | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- noMediaPresent - No media present in a position where it should be or the media was removed during the operation.
- pageTurnFail - The device was not able to turn the page.
- mediaJammed - The media is jammed. Operator intervention is required.
- passbookClosed - There were fewer pages left than specified to turn. As a result of the operation, the passbook has been closed.
- lastOrFirstPageReached - The printer cannot close the passbook because there were fewer pages left than specified to turn. As a result of the operation, the last or the first page has been reached and is open.
- mediaSize - The media has an incorrect size.

Event Messages

None

[11.3.21 - Printer.SetBlackMarkMode](#)

This command switches the black mark detection mode and associated functionality on or off. The black mark detection mode is persistent. If the selected mode is already active this command will complete with success.

Command Message

| Payload | Type | Required |
|-----------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "blackMarkMode": "on" | string | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

blackMarkMode

Specifies the desired black mark detection mode as one of the following:

- on - Turns the black mark detection and associated functionality on.
- off - Turns the black mark detection and associated functionality off.

Completion Message

| Payload | Type | Required |
|---|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

Event Messages

None

11.4 - Event Messages

11.4.1 - Printer.MediaPresentedEvent

This event is used to indicate when media has been presented to the customer for removal.

| Payload | Type | Required |
|------------------------|------|----------|
| { | | |
| "wadIndex": 0, integer | | |
| "totalWads": 0 integer | | |
| } | | |

Properties

wadIndex

Specifies the index (starting from one) of the presented wad, where a Wad is a bunch of one or more pages presented as a bunch.

totalWads

Specifies the total number of wads in the print job, zero if the total number of wads is not known.

11.4.2 - Printer.NoMediaEvent

This event specifies that the physical media must be inserted into the device in order for the execute command to proceed.

| Payload | Type | Required |
|--|------|----------|
| { | | |
| "userPrompt": Add example to YAML string | | |
| } | | |

Properties

userPrompt

The user prompt from the form definition. This will be omitted if either a form does not

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

define a value for the user prompt or the event is being generated as the result of a command that does not use forms.

The client may use the this in any manner it sees fit, for example it might display the string to the operator, along with a message that the media should be inserted.

[11.4.3 - Printer.MediaInsertedEvent](#)

This event specifies that the physical media has been inserted into the device.

The client may use this event to, for example, remove a message box from the screen telling the user to insert media.

[11.4.4 - Printer.FieldErrorEvent](#)

This event specifies that a fatal error has occurred while processing a field.

| Payload | Type | Required |
|-----------------------------------|--------|----------|
| { | | |
| "formName": Add example to YAML, | string | |
| "fieldName": Add example to YAML, | string | |
| "failure": "required" | string | |
| } | | |

Properties

formName

The form name.

fieldName

The field name.

failure

Specifies the type of failure as one of the following:

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

- required - The specified field must be supplied by the client.
- staticOverwrite - The specified field is static and thus cannot be overwritten by the client.
- overflow - The value supplied for the specified fields is too long.
- notFound - The specified field does not exist.
- notRead - The specified field is not an input field.
- notWrite - An attempt was made to write to an input field.
- hwerror - The specified field uses special hardware (e.g. OCR, Low/High coercivity, etc) and an error occurred.
- notSupported - The form field type is not supported with device.
- graphic - The specified graphic image could not be printed.

[11.4.5 - Printer.FieldWarningEvent](#)

This event specifies that a non-fatal error has occurred while processing a field.

| Payload | Type | Required |
|-----------------------------------|--------|----------|
| { | | |
| "formName": Add example to YAML, | string | |
| "fieldName": Add example to YAML, | string | |
| "failure": "required" | string | |
| } | | |

Properties

formName

The form name.

fieldName

The field name.

failure

Specifies the type of failure as one of the following:

- required - The specified field must be supplied by the client.

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

- staticOverwrite - The specified field is static and thus cannot be overwritten by the client.
- overflow - The value supplied for the specified fields is too long.
- notFound - The specified field does not exist.
- notRead - The specified field is not an input field.
- notWrite - An attempt was made to write to an input field.
- hwerror - The specified field uses special hardware (e.g. OCR, Low/High coercivity, etc) and an error occurred.
- notSupported - The form field type is not supported with device.
- graphic - The specified graphic image could not be printed.

[11.4.6 - Printer.MediaRejectedEvent](#)

This event is generated as a result of physical media that is rejected whenever an attempt is made to insert media into the physical device. Rejection of the media will cause the command currently executing to complete with an error, at which point the media should be removed.

The client may use this event to (for example) display a message box on the screen indicating why the media was rejected, and telling the user to remove and reinsert the media.

| Payload | Type | Required |
|---------------------------|------|----------|
| { | | |
| "reason": "short" string | | |
| } | | |

Properties

reason

Specifies the reason for rejecting the media as one of the following values:

- short - The rejected media was too short.
- long - The rejected media was too long.
- multiple - The media was rejected due to insertion of multiple documents.
- align - The media could not be aligned and was rejected.
- moveToAlign - The media could not be transported to the align area and was rejected.
- shutter - The media was rejected due to the shutter failing to close.

- escrow - The media was rejected due to problems transporting media to the escrow position.
- thick - The rejected media was too thick.
- other - The media was rejected due to a reason other than those listed above.

11.5 - Unsolicited Messages

11.5.1 - Printer.MediaTakenEvent

This event is sent when the media is taken from the exit slot following the completion of a successful eject operation or following a [Printer.MediaRejectedEvent](#). For devices that do not physically move media, this event may also be generated when the media is taken from the device.

11.5.2 - Printer.MediaInsertedUnsolicitedEvent

This event specifies that the physical media has been inserted into the device without any read or print execute commands being executed. This event is only generated when media is entered in an unsolicited manner.

11.5.3 - Printer.MediaPresentedUnsolicitedEvent

This event is used to indicate when media has been presented to the customer for removal as a result of a print operation through some non XFS interface.

| Payload | Type | Required |
|----------------|---------|----------|
| { | | |
| "wadIndex": 0, | integer | |
| "totalWads": 0 | integer | |
| } | | |

Properties

wadIndex

Specifies the index (starting from one) of the presented wad, where a Wad is a bunch of one

or more pages presented as a bunch.

totalWads

Specifies the total number of wads in the print job, zero if the total number of wads is not known.

[11.5.4 - Printer.MediaDetectedEvent](#)

This event is generated when a media is detected in the device during a reset operation.

| Payload | Type | Required |
|--------------------------|---------|----------|
| { | | |
| "position": "retracted", | string | |
| "retractBinNumber": 0 | integer | |
| } | | |

Properties

position

Specifies the media position after the reset operation, as one of the following values:

- retracted - The media was retracted during the reset operation.
- present - The media is in the print position or on the stacker.
- entering - The media is in the exit slot.
- jammed - The media is jammed in the device.
- unknown - The media is in an unknown position.
- expelled - The media was expelled during the reset operation.

retractBinNumber

Number of the retract bin the media was retracted to. This number has to be between one and the [number of bins](#) supported by this device. It is only relevant if [position](#) is *retracted*.

11.5.5 - Printer.RetractBinStatusEvent

This event specifies that the status of the retract bin holding the retracted media has changed.

| Payload | Type | Required |
|---------------------|---------|----------|
| { | | |
| "binNumber": 0, | integer | |
| "state": "inserted" | string | |
| } | | |

Properties

binNumber

Number of the retract bin for which the status has changed.

state

Specifies the current state of the retract bin as one of the following values:

- inserted - The retract bin has been inserted.
- removed - The retract bin has been removed.

11.5.6 - Printer.DefinitionLoadedEvent

This event is used to indicate when a form or media definition has successfully been loaded via the [Printer.LoadDefinition](#) command.

| Payload | Type | Required |
|------------------------------|--------|----------|
| { | | |
| "name": Add example to YAML, | string | |
| "type": "form" | string | |
| } | | |

Properties

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

name

Specifies the name of the form or media just loaded.

type

Specifies the type of definition loaded. This field can be one of the following values:

- **form** - The form identified by **name** has been loaded.
- **media** - The media identified by **name** has been loaded.

[11.5.7 - Printer.MediaAutoRetractedEvent](#)

This event indicates when media has been automatically retracted by the device. Support for this event is indicated when **autoRetractPeriod** is non-zero. The event can be generated as the result of any command that presents media to the customer.

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "retractResult": "ok" , string | | |
| "binNumber": 0 | integer | |
| } | | |

Properties

retractResult

Specifies the result of the automatic retraction, as one of the following values:

- **ok** - The media was retracted successfully.
- **jammed** - The media is jammed.

binNumber

Number of the retract bin the media was retracted to or zero if the media is retracted to the transport. This number has to be between zero and the number of bins supported by this device. This value is also zero if **retractResult** is *jammed*.

11.5.8 - Printer.RetractBinThresholdEvent

This event specifies that the status of the retract bin holding the retracted media has changed.

| Payload | Type | Required |
|-----------------|---------|----------|
| { | | |
| "binNumber": 0, | integer | |
| "state": "ok" | string | |
| } | | |

Properties

binNumber

Number of the retract bin for which the status has changed.

state

Specifies the current state of the retract bin as one of the following:

- ok - The retract bin of the printer is in a good state.
- full - The retract bin of the printer is full.
- high - The retract bin of the printer is high.

11.5.9 - Printer.PaperThresholdEvent

This user event is used to specify that the state of the paper reached a threshold. There is no threshold defined for the parking station as this can contain only one paper item.

| Payload | Type | Required |
|-------------------------|--------|----------|
| { | | |
| "paperSource": "upper", | string | |
| "threshold": "full" | string | |
| } | | |

Properties

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

paperSource

Specifies the paper source as one of the following:

- upper - The only paper source or the upper paper source, if there is more than one paper supply.
- lower - The lower paper source.
- external - The external paper source (such as envelope tray or single sheet feed).
- aux - The auxiliary paper source.
- aux2 - The second auxiliary paper source.

threshold

Specifies the current state of the paper source as one of the following:

- full - The paper in the paper source is in a good state.
- low - The paper in the paper source is low.
- out - The paper in the paper source is out.

[11.5.10 - Printer.TonerThresholdEvent](#)

This user event is used to specify that the state of the toner (or ink) reached a threshold.

| Payload | Type | Required |
|-----------------|--------|----------|
| { | | |
| "state": "full" | string | |
| } | | |

Properties**state**

Specifies the current state of the toner (or ink) as one of the following:

- full - The toner (or ink) in the printer is in a good state.
- low - The toner (or ink) in the printer is low.
- out - The toner (or ink) in the printer is out.

11.5.11 - Printer.LampThresholdEvent

This user event is used to specify that the state of the imaging lamp reached a threshold.

| Payload | Type | Required |
|---------------|--------|----------|
| { | | |
| "state": "ok" | string | |
| } | | |

Properties

state

Specifies the current state of the imaging lamp as one of the following values:

- ok - The imaging lamp is in a good state.
- fading - The imaging lamp is fading and should be changed.
- inop - The imaging lamp is inoperative.

11.5.12 - Printer.InkThresholdEvent

This user event is used to specify that the state of the stamping ink reached a threshold.

| Payload | Type | Required |
|-----------------|--------|----------|
| { | | |
| "state": "full" | string | |
| } | | |

Properties

state

Specifies the current state of the stamping ink as one of the following:

- full - The stamping ink in the printer is in a good state.
- low - The stamping ink in the printer is low.
- out - The stamping ink in the printer is out.

12 - Text Terminal Interface

This chapter defines the Text Terminal interface functionality and messages.

12.1 - Summary

This section describes the functions provided by a generic Text Terminal Unit () service. A Text Terminal Unit is a text i/o device, which applies both to ATM operator panels and to displays incorporated in devices such as pads and printers. This service allows for the following categories of functions:

- Forms oriented input and output
- Direct display output
- Keyboard input
- LED settings and control

All position indexes are zero based, where column zero, row zero is the top-leftmost position.

If the device has no shift key, the [TextTerminal.ReadForm](#) and [TextTerminal.Read](#) commands will return only upper case letters. If the device has a shift key, these commands return upper and lower case letters as governed by the user's use of the shift key.

12.2 - General Information

12.2.1 - Form and Field Definitions

This section outlines the format of the definitions of forms, the fields within them, and the media on which they are printed.

Definition Syntax

The syntactic rules for form, field and media definitions are as follows:

- **White space**
space, tab
- **Line continuation**
backslash (\)

- **Line termination**

CR, LF, CR/LF; line termination ends a "keyword section" (a keyword and its value[s])

- **Keywords**

must be all upper case

- **Names**

(field/media/font names) any case; case is preserved; Service Providers are case sensitive.

- **Strings**

all strings must be enclosed in double quote characters ("); standard C escape sequences are allowed.

- **Comments**

start with two forward slashes (//); end at line termination.

Other notes:

- If a keyword is present, all its values must be specified; default values are used only if the keyword is absent.
- Values that are character strings are marked with asterisks in the definitions below, and must be quoted as specified above.
- Fields are processed in the sequence they are defined in the form.
- The order of attributes within a form is not mandatory; the attributes may be defined in any order.
- All forms can be represented using either ISO 646 (ANSI) or unicode character encoding. If the unicode representation is used then all Names and Strings are restricted to an internal representation of ISO 646 (ANSI) characters. Only the initialValue keyword values can have double byte values outside of the ISO 646 (ANSI) character set.
- If forms character encoding is unicode then, consistent with the unicode standard, the file prefix must be in Little Endian (xFFFE) or Big Endian (xFEFF) notation, such that unicode encoding is recognized.
- In the form definition file, where characters are expressed using standard C hexadecimal escape sequences, the high order byte is defined first. For example, "\x0041" would represent the character 'A'. This is independent of the encoding format of the form definition file

Form/media definition files in multi-vendor environments

Although for most Service Providers directory location and extension of form/media definition files are configurable through the registry, the capabilities of Service Providers and or actual hardware may vary. Therefore the following considerations should be taken into account when clients use form definition files with the purpose of running in a multi-vendor environment:

- Physical display area dimensions may vary from one text terminal to another.

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

- Just-in-time form loading may not be supported by all Service Providers, which makes it impossible to create dynamic form files just before displaying them (which in return means that only the display data of the forms can be changed, not the layout data such as field positions).
- Some form/media definition keywords may not be supported due to limitations of the hardware or software.

Form Definition

| | | |
|----------------------------|-------------------------------------|--|
| FORM | <i>formname*</i> | |
| BEGIN | | |
| (required) SIZE | <i>width height</i> | Width of form Height of form |
| VERSION | <i>major, minor, date*, author*</i> | Major version number (default 0) Minor version number (default 0) Creation/modification date Author of form |
| (required) LANGUAGE | <i>languageID</i> | Language used in this form - a 16 bit value (LANGID) which is a combination of a primary (10 bits) and a secondary (6 bits) language ID (This is the standard language ID in the Win32 API; standard macros support construction and decomposition of this composite ID) |
| COPYRIGHT | <i>copyright*</i> | Copyright entry |
| TITLE | <i>title*</i> | Title of form |
| COMMENT | <i>comment*</i> | Comment section |
| [FIELD | <i>fieldname*</i> | One field definition (as defined in the next section) for each field in the form |
| BEGIN ... | | |
| END] | | |
| END | | |

Field Definition

| | | |
|-----------------|-------------------|--|
| FIELD | <i>fieldname*</i> | |
| BEGIN | | |
| LANGUAGE | <i>languageID</i> | Language used for this field. See Form definition for detailed description. If unspecified defaults to form definition |

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

| | | |
|----------------------------|----------------------|---|
| | | Language specification. |
| (required) POSITION | <i>x, y</i> | Horizontal position (relative to left side of form) Vertical position (relative to top of form) The initial left upper position is referenced as (0,0) |
| (required) SIZE | <i>width, height</i> | Field width Field height |
| TYPE | <i>fieldtype</i> | Type of field: Text (default) Invisible Password (contents is echoed with '*') Graphic (ignored for TextTerminal.ReadForm commands) |
| SCALING | <i>scalingtype</i> | Information on how to size the Graphic within the field: BestFit (default) scale to size indicated ASIS render at native size Maintain aspect scale as close as possible to size indicated while maintaining the aspect ratio and not losing Graphic information. SCALING is only relevant for Graphics field types |
| CLASS | <i>class</i> | Field class: Optional (default) Static Required |
| KEYS | <i>keys</i> | Accepted input key types: Numeric Hexadecimal Alphanumeric This is an optional field where the default value is vendor dependent. |
| ACCESS | <i>access</i> | Access rights of field: Write (default) Read ReadWrite |
| OVERFLOW | <i>overflow</i> | Action on field overflow: Terminate (default) Truncate OverWrite |
| STYLE | <i>style</i> | Display attributes as a combination of the following, ORed together using the " |
| HORIZONTAL | <i>justify</i> | Horizontal alignment of field contents: Left (default) Right Center |
| FORMAT | <i>formatstring*</i> | This is a client defined input field describing how the client should format the data. This may be interpreted by the Service Provider. |
| INITIALVALUE | <i>value*</i> | Initial value. For Graphic type fields, this value will contain the filename of the Graphic image. The type of this Graphic will be determined by the file extension (e.g. BMP for Windows Bitmap). The Graphic file name must contain the full path. For example "C:\XFS\BSVCLOGO.BMP" illustrates the use of the full path name |

END

12.3 - Command Messages

12.3.1 - TextTerminal.GetFormList

This command is used to retrieve the list of forms available on the device.

Command Message

| Payload | Type | Required |
|--------------------------------------|------|----------|
| { "timeout": 5000 integer } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "formList": [Add example to YAML] | array (string) | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

formList

Array of the form names.

Event Messages

None

[12.3.2 - TextTerminal.GetQueryForm](#)

This command is used to retrieve details of the definition of a specified form.

Command Message

| Payload | Type | Required |
|---------------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "formName": Add example to YAML | string | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not

timeout but can be cancelled.

default: 0

formName

Contains the form name on which to retrieve details.

Completion Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "formName": { | object | |
| }, | | |
| "width": Add example to YAML, | string | |
| "height": Add example to YAML, | string | |
| "versionMajor": Add example to YAML, | string | |
| "versionMinor": Add example to YAML, | string | |
| "charSupport": "ascii", | string | |
| "fields": [Add example to YAML], | array (string) | |
| "languageId": Add example to YAML | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

formName

Specifies the null-terminated name of the form.

width

Specifies the width of the form in columns.

height

Specifies the height of the form in rows.

versionMajor

Specifies the major version. If version is not specified in the form then zero is returned.

versionMinor

Specifies the minor version. If the version is not specified in the form then zero is returned.

charSupport

A single flag indicating whether the form is encoded in ascii or unicode.

fields

Object to a list of the field names.

languageId

Specifies the language identifier for the form.

Event Messages

None

12.3.3 - TextTerminal.GetQueryField

This command is used to retrieve details of the definition of a single or all fields on a specified form.

Command Message

| Payload | Type | Required |
|----------------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "formName": Add example to YAML, | string | |
| "fieldName": Add example to YAML | string | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

formName

Specifies the form name

fieldName

Specifies the name of the field about which to retrieve details. If this value is not set, then retrieve details for all fields on the form.

Completion Message

| Payload | Type | Required |
|---------|------|----------|
| { | | |

```

"completionCode": "success",           string
"errorDescription": Add example to YAML, string
"fields": [{                           array (object)
  "fieldName": Add example to YAML,     string
  "type": "text",                      string
  "class": "static",                   string
  "access": Add example to YAML,       string
  "read": Add example to YAML,         string
  "write": Add example to YAML,        string
  "overflow": "terminate",             string
  "format": Add example to YAML,       string
  "languageId": Add example to YAML,   string
}],                                 string
"formName": ,                         undefined
"width": ,                            undefined
"height": ,                           undefined
"versionMajor": ,                     undefined
"versionMinor": ,                     undefined
"charSupport": ,                     undefined
"languageId": ,                      undefined
}

```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

fields

Array of Fields.

fields/fieldName

Specifies the field name.

fields/type

Specifies the type of field.

fields/class

Specifies the class of the field.

fields/access

Specifies whether the field is to be used for input, output or both.

fields/access/read

The Field is used for input from the physical device.

fields/access/write

The Field is used for output to the physical device.

fields/overflow

Specifies how an overflow of field data should be handled.

fields/format

Format string as defined in the form for this field.

fields/languageId

Specifies the language identifier for the field.

Event Messages

None

[12.3.4 - TextTerminal.GetKeyDetail](#)

This command returns information about the Keys (buttons) supported by the device. This command should be issued to determine which Keys are available.

Command Message

| Payload | Type | Required |
|-----------------|---------|----------|
| { | | |
| "timeout": 5000 | integer | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|------------------------------|--------|----------|
| { | | |
| "completionCode": "success", | string | |

```
"errorDescription": Add example to YAML,  string  
"keys": Add example to YAML,          string  
"commandKeys": [Add example to YAML]  array (string)  
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

keys

String which holds the printable characters (numeric and alphanumeric keys) on the Text Terminal Unit, e.g. “0123456789ABCabc” if those text terminal input keys are present. This field is not set if no keys of this type are present on the device.

commandKeys

Array of command keys on the Text Terminal Unit.

Event Messages

None

[12.3.5 - TextTerminal.Beep](#)

This command is used to beep at the text terminal unit.

Command Message

| Payload | Type | Required |
|-----------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "beep": { | object | |
| "off": false, | boolean | |
| "keyPress": false, | boolean | |
| "exclamation": false, | boolean | |
| "warning": false, | boolean | |
| "error": false, | boolean | |
| "critical": false, | boolean | |
| "continuous": false | boolean | |
| } | | |
| } | | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

beep

Specifies whether the beeper should be turned on or off.

beep/off

The beeper is turned off.

beep/keyPress

The beeper sounds a key click signal.

beep/exclamation

The beeper sounds an exclamation signal.

beep/warning

The beeper sounds a warning signal.

beep/error

The beeper sounds a error signal.

beep/critical

The beeper sounds a critical error signal.

beep/continuous

The beeper sound is turned on continuously.

Completion Message

| Payload | Type | Required |
|---|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

Event Messages

None

12.3.6 - TextTerminal.ClearScreen

This command clears the specified area of the text terminal unit screen. The cursor is positioned to the upper left corner of the cleared area.

Command Message

| Payload | Type | Required |
|------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "positionX": 0, | integer | |
| "positionY": 0, | integer | |
| "width": 0, | integer | |
| "height": 0 | integer | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

positionX

Specifies the horizontal position of the area to be cleared.

positionY

Specifies the vertical position of the area to be cleared.

width

Specifies the width position of the area to be cleared.

height

Specifies the height position of the area to be cleared.

Completion Message

| Payload | Type | Required |
|---|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

Event Messages

None

12.3.7 - TextTerminal.DispLight

This command is used to switch the lighting of the text terminal unit on or off.

Command Message

| Payload | Type | Required |
|--------------------------------------|------|----------|
| { "timeout": 5000 integer } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "mode": false | boolean | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

mode

Specifies whether the lighting of the text terminal unit is switched on (TRUE) or off (FALSE).

Event Messages

None

12.3.8 - TextTerminal.SetLed

This command is used to set the status of the LEDs.

Command Message

| Payload | Type | Required |
|-----------------------|-------------|-----------------|
| { | | |
| "timeout": 5000, | integer | |
| "led": 0, | integer | |
| "command": { | object | |
| "off": false, | boolean | |
| "slowFlash": false, | boolean | |
| "mediumFlash": false, | boolean | |
| "quickFlash": false, | boolean | |
| "continuous": false, | boolean | |
| "red": false, | boolean | |
| "green": false, | boolean | |

```
"yellow": false,      boolean
"blue": false,        boolean
"cyan": false,        boolean
"magenta": false,    boolean
"white": false,       boolean
}
}
```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

led

Specifies the index array as reported in Capabilities of the LED to set as one of the values defined within the capabilities section [TextTerminal.Capabilities](#)

command

Specifies off type A or a combination of the following flags consisting of one type B, and optionally one type C. If no value of type C is specified then the default color is used. The Service Provider determines which color is used as the default color.

command/off

The LED is turned off. Type A

command/slowFlash

The LED is set to flash slowly. Type B

command/mediumFlash

The LED is set to flash medium frequency. Type B

command/quickFlash

The LED is set to flash quickly. Type B

command/continuous

The LED is turned on continuously(steady). Type B

command/red

The LED color is set to red. Type C

command/green

The LED color is set to green. Type C

command/yellow

The LED color is set to yellow. Type C

command/blue

The LED color is set to blue. Type C

command/cyan

The LED color is set to cyan. Type C

command/magenta

The LED color is set to magenta. Type C

command/white

The LED is set to white. Type C

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "invalidLed" | string | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- invalidLed - An attempt to set a LED to a new value was invalid because the LED does not exist.

Event Messages

None

12.3.9 - TextTerminal.SetResolution

This command is used to set the resolution of the display. The screen is cleared and the cursor is positioned at the upper left position.

Command Message

| Payload | Type | Required |
|------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "resolution": { | object | |
| "sizeX": 0, | integer | |
| "sizeY": 0 | integer | |
| } | | |
| } | | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

resolution

Specifies the horizontal size of the display of the text terminal unit.

resolution/sizeX

Specifies the horizontal size of the display of the text terminal unit (the number of columns that can be displayed).

resolution/sizeY

Specifies the vertical size of the display of the text terminal unit (the number of rows that can be displayed).

Completion Message

| Payload | Type | Required |
|---------|------|----------|
|---------|------|----------|

```
{
  "completionCode": "success",           string
  "errorDescription": Add example to YAML, string
  "errorCode": "resolutionNotSupported"   string
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- resolutionNotSupported - The specified resolution is not supported by the display.

Event Messages

None

12.3.10 - TextTerminal.WriteForm

This command is used to display a form by merging the supplied variable field data with the defined form and field data specified in the form.

Command Message

| Payload | Type | Required |
|---------|------|----------|
| { | | |

```

"timeout": 5000,           integer
"formName": Add example to YAML, string
"clearScreen": false,      boolean
"fields": [Add example to YAML] array (string)
}

```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

formName

Form name.

clearScreen

Specifies whether the screen is cleared before displaying the form (TRUE) or not (FALSE).

fields

Specifies "=" string. e.g. Field1=123. The stands for a string containing all the printable characters (numeric and alphanumeric) to display on the text terminal unit key pad for this field.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |

```
"errorCode": "formNotFound"           string
{
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- formNotFound - The specified form definition cannot be found.
- formInvalid - The specified form definition is invalid.
- mediaOverflow - The form overflowed the media.
- fieldSpecFailure - The syntax of the lpszFields member is invalid.
- characterSetsData - Character set(s) supported by Service Provider is inconsistent with use of fields value.
- fieldError - An error occurred while processing a field.

Event Messages

None

[12.3.11 - TextTerminal.ReadForm](#)

This command is used to read data from input fields on the specified form.

Command Message

| Payload | Type | Required |
|---------|------|----------|
| { | | |

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

```

"timeout": 5000,           integer
"formName": Add example to YAML,   string
"fieldNames": [Add example to YAML] array (string)
}

```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

formName

Specifies the null-terminated name of the form

fieldNames

Specifies the field names from which to read input data. The fields are edited by the user in the order that the fields are specified within this parameter. If fieldNames value is not set, then data is read from all input fields on the form in the order they appear in the form file (independent of the field screen position).

Completion Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "formNotFound", | string | |
| "fields": [Add example to YAML] | array (string) | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- formNotFound - The specified form definition cannot be found.
- formInvalid - The specified form definition is invalid.
- fieldSpecFailure - The syntax of the lpszFields member is invalid.
- keyCanceled - The read operation was terminated by pressing the key.
- fieldError - An error occurred while processing a field.

fields

Specifies "=" string. e.g. Field1=123. This stands for a string containing all the printable characters (numeric and alphanumeric) read from the text terminal unit key pad for this field.

Event Messages

None

12.3.12 - TextTerminal.Write

This command displays the specified text on the display of the text terminal unit. The specified text may include the control characters CR (Carriage Return) and LF (Line Feed). The control characters can be included in the text as CR, or LF, or CR LF, or LF CR and all combinations will perform the function of relocating the cursor position to the left hand side of the display on the next line down. If the text will overwrite the display area then the display will scroll.

Command Message

| Payload | Type | Required |
|-----------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "mode": "relative", | string | |
| "posX": 0, | integer | |
| "posY": 0, | integer | |
| "textAttr": { | object | |
| "underline": false, | boolean | |
| "inverted": false, | boolean | |
| "flash": false | boolean | |
| }, | | |
| "text": Add example to YAML | string | |
| } | | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

mode

Specifies whether the position of the output is absolute or relative to the current cursor position.

posX

If mode is set to absolute, this specifies the absolute horizontal position. If mode is set to relative this specifies a horizontal offset relative to the current cursor position as a zero (0) based value.

posY

If mode is set to absolute, this specifies the absolute vertical position. If mode is set to relative this specifies a vertical offset relative to the current cursor position as a zero (0) based value.

textAttr

Specifies the text attributes used for displaying the text. If none of the following attribute flags are selected then the text will be displayed as normal text.

textAttr/underline

The displayed text will be unlined.

textAttr/inverted

The displayed text will be inverted.

textAttr/flash

The displayed text will be flashing.

text

Specifies the text that will be displayed.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "characterSetsData" | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- characterSetsData - Character set(s) supported by Service Provider is inconsistent with use of text value.

Event Messages

None

12.3.13 - TextTerminal.Read

This command activates the keyboard of the text terminal unit for input of the specified number of characters. Depending on the specified flush mode the input buffer is cleared. During this command, pressing an active key results in a [TextTerminal.KeyEvent](#) event containing the key details. On completion of the command (when the maximum number of keys have been pressed or a terminator key is pressed), the entered string, as interpreted by the Service Provider, is returned. The Service Provider takes command keys into account when interpreting the data.

Command Message

| Payload | Type | Required |
|---------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "numOfChars": 0, | integer | |
| "mode": "relative", | string | |

```

"posX": 0,                                integer
"posY": 0,                                integer
"echoMode": "text",                         string
"echoAttr": {                               object
    "underline": false,                     boolean
    "inverted": false,                     boolean
    "flash": false,                        boolean
},
"echo": false,                             boolean
"flush": false,                           boolean
"autoEnd": false,                          boolean
"activeKeys": Add example to YAML,          string
"activeCommandKeys": [Add example to YAML], array (string)
"terminateCommandKeys": [Add example to YAML] array (string)
}

```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

numOfChars

Specifies the number of printable characters (numeric and alphanumeric keys) that will be read from the text terminal unit key pad. All command keys like ckEnter, ckFDK01 will not be counted.

mode

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

Specifies where the cursor is positioned for the read operation.

posX

If mode is set to absolute, this specifies the absolute horizontal position. If mode is set to relative this specifies a horizontal offset relative to the current cursor position as a zero (0) based value.

posY

If mode is set to absolute, this specifies the absolute vertical position. If mode is set to relative this specifies a vertical offset relative to the current cursor position as a zero (0) based value.

echoMode

Specifies how the user input is echoed to the screen.

echoAttr

Specifies the text attributes with which the user input is echoed to the screen. If none of the following attribute flags are selected then the text will be displayed as normal text.

echoAttr/underline

The displayed text will be underlined.

echoAttr/inverted

The displayed text will be inverted.

echoAttr/flash

The displayed text will be flashing.

echo

Specifies whether the cursor is visible(TRUE) or invisible(FALSE).

flush

Specifies whether the keyboard input buffer is cleared before allowing for user input(TRUE) or not (FALSE).

autoEnd

Specifies whether the command input is automatically ended by Service Provider if the maximum number of printable characters as specified with numOfChars is entered.

activeKeys

String which specifies the numeric and alphanumeric keys on the Text Terminal Unit, e.g. "12ABab", to be active during the execution of the command. Devices having a shift key interpret this parameter differently from those that do not have a shift key. For devices having a shift key, specifying only the upper case of a particular letter enables both upper and lower case of that key, but the device converts lower case letters to upper case in the output parameter. To enable both upper and lower case keys, and have both upper and lower case letters returned, specify both the upper and lower case of the letter (e.g. "12AaBb"). For devices not having a shift key, specifying either the upper case only (e.g. "12AB"), or specifying both the upper and lower case of a particular letter (e.g. "12AaBb"), enables that key and causes the device to return the upper case of the letter in the output parameter. For both types of device, specifying only lower case letters (e.g. "12ab") produces a key invalid error. This parameter is a NULL if no keys of this type are active keys. activeKeys and activeUnicodeKeys are mutually exclusive, so activeKeys field must not be set if activeUnicodeKeys field is not set.

activeCommandKeys

Array specifying the command keys which are active during the execution of the command. The array is terminated with a zero value and this array is not set if no keys of this type are active keys.

terminateCommandKeys

Array specifying the command keys which must terminate the execution of the command. The array is terminated with a zero value and this array is not set if no keys of this type are terminate keys.

Completion Message

| <u>Payload</u> | <u>Type</u> | <u>Required</u> |
|----------------|-------------|-----------------|
|----------------|-------------|-----------------|

```
{  
  "completionCode": "success",           string  
  "errorDescription": Add example to YAML, string  
  "errorCode": "keyInvalid",            string  
  "input": Add example to YAML          string  
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- keyInvalid - At least one of the specified keys is invalid.
- keyNotSupported - At least one of the specified keys is not supported by the Service Provider.
- noActiveKeys - There are no active keys specified.

input

Specifies a zero terminated string containing all the printable characters (numeric and alphanumeric) read from the text terminal unit key pad.

Event Messages

None

12.3.14 - TextTerminal.Reset

Sends a service reset to the Service Provider. This command clears the screen, clears the keyboard buffer, sets the default resolution and sets the cursor position to the upper left.

Command Message

| Payload | Type | Required |
|-----------------|---------|----------|
| { | | |
| "timeout": 5000 | integer | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|---|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

Event Messages

None

[12.3.15 - TextTerminal.DefineKeys](#)

This command defines the keys that will be active during the next [TextTerminal.ReadForm](#) command. The configured set will be active until the next [TextTerminal.ReadForm](#) command ends, at which point the default values are restored.

Command Message

| Payload | Type | Required |
|---|----------------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "activeKeys": Add example to YAML, | string | |
| "activeCommandKeys": [Add example to YAML], | array (string) | |
| "terminateCommandKeys": [Add example to YAML] | array (string) | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

activeKeys

String which specifies the alphanumeric keys on the Text Terminal Unit, e.g. "12ABab", to be active during the execution of the next [TextTerminal.ReadForm](#) command. Devices having a shift key interpret this parameter differently from those that do not have a shift

key. For devices having a shift key, specifying only the upper case of a particular letter enables both upper and lower case of that key, but the device converts lower case letters to upper case in the output parameter. To enable both upper and lower case keys, and have both upper and lower case letters returned, specify both the upper and lower case of the letter (e.g. "12AaBb"). For devices not having a shift key, specifying either the upper case only (e.g. "12AB"), or specifying both the upper and lower case of a particular letter (e.g. "12AaBb"), enables that key and causes the device to return the upper case of the letter in the output parameter. For both types of device, specifying only lower case letters (e.g. "12ab") produces a key invalid error.

activeCommandKeys

Array specifying the command keys which are active during the execution of the next [TextTerminal.ReadForm](#) command.

terminateCommandKeys

Array specifying the command keys which must terminate the execution of the next [TextTerminal.ReadForm](#) command.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "keyInvalid" | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

information

errorCode

Specifies the error code if applicable. The following values are possible:

- keyInvalid - At least one of the specified keys is invalid.
- keyNotSupported - At least one of the specified keys is not supported by the Service Provider.
- noActiveKeys - There are no active keys specified.

Event Messages

None

12.4 - Unsolicited Messages

12.4.1 - TextTerminal.FieldErrorEvent

This event specifies that a fatal error has occurred while processing a field.

| Payload | Type | Required |
|-----------------------------------|--------|----------|
| { | | |
| "formName": Add example to YAML, | string | |
| "fieldName": Add example to YAML, | string | |
| "failure": "required" | string | |
| } | | |

Properties

formName

Specifies the form name.

fieldName

Specifies the field name.

failure

Specifies the type of failure.

[12.4.2 - TextTerminal.FieldWarningEvent](#)

This event is used to specify that a non-fatal error has occurred while processing a field.

[12.4.3 - TextTerminal.KeyEvent](#)

This event specifies that any active key has been pressed at the TTU during the [TextTerminal.Read](#) command. In addition to giving the client more details about individual key presses this information may also be used if the device has no internal display unit and the client has to manage the display of the entered digits.

| Payload | Type | Required |
|-----------------------------------|--------|----------|
| { | | |
| "key": Add example to YAML, | string | |
| "commandKey": Add example to YAML | string | |
| } | | |

Properties

key

On a numeric or alphanumeric key press this parameter holds the value of the key pressed. This value is not set if no numeric or alphanumeric key was pressed.

commandKey

On a Command key press this parameter holds the value of the Command key pressed, e.g. ckEnter. This value is not set when no command key was pressed.

13 - Sensors and Indicators Interface

This chapter defines the Sensors and Indicators interface functionality and messages.

13.1 - Summary

TODO

13.2 - Command Messages

13.2.1 - SensorsAndIndicators.SetGuidanceLight

This command is used to set the status of the devices guidance lights. This includes defining the flash rate, the color and the direction. When a client tries to use a color or direction that is not supported then the Service Provider will return the generic completionCode unsupportedData.

Command Message

| Payload | Type | Required |
|----------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "guidLight": 0, | integer | |
| "command": { | object | |
| "flashRate": "off", | string | |
| "color": "default", | string | |
| "direction": "entry" | string | |
| } | | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

guidLight

Specifies the index of the guidance light to set as one of the values defined within the capabilities section:

command/flashRate

Indicates which flash rates are supported by the guidelight.

command/color

Indicates which colors are supported by the guidelight.

command/direction

Indicates which directions are supported by the guidelight. and it's an optional field

Completion Message

| Payload | Type | Required |
|---|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML | string | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

Event Messages

None

13.2.2 - SensorsAndIndicators.GetAutoStartupTime

This command is used to retrieve the availability of the auto start-up time function as well as the current configuration of the auto start-up time.

Command Message

| Payload | Type | Required |
|--------------------------------------|------|----------|
| { "timeout": 5000 integer } | | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|---|------|----------|
| { "completionCode": "success", string | | |

```
"errorDescription": Add example to YAML,  string
"errorCode": "invalidPort",          string
"wMode": ,                         undefined
"startTime": ,                      undefined
"year": ,                          undefined
"month": ,                         undefined
"dayOfWeek": ,                     undefined
"day": ,                           undefined
"hour": ,                          undefined
"minute": ,                        undefined
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable.

Event Messages

None

13.2.3 - SensorsAndIndicators.ClearAutoStartupTime

This command is used to clear the time at which the machine will automatically start.

Command Message

| Payload | Type | Required |
|--------------------------------------|------|----------|
| { "timeout": 5000 integer } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|------|----------|
| { "completionCode": "success", string "errorDescription": Add example to YAML string } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

information

Event Messages

None

13.2.4 - SensorsAndIndicators.Register

This command is used to register for, or deregister events from the Sensors and Indicators Unit. The default condition is that all events are deregistered. The events are only registered or deregistered for the session which sends the command, all other sessions are unaffected.

No action has been taken if this command returns an error. If a hardware error occurs while executing the command, the command will return OK, but event(s) will be generated which indicate(s) the port(s) which have failed.

Command Message

| Payload | Type | Required |
|-----------------------------------|--------|----------|
| { | | |
| "operatorSwitch": "register", | string | |
| "tamperSensor": "register", | string | |
| "intTamperSensor": "register", | string | |
| "seismicSensor": "register", | string | |
| "heatSensor": "register", | string | |
| "proximitySensor": "register", | string | |
| "ambientLightSensor": "register", | string | |
| "enhancedAudio": "register", | string | |
| "bootSwitch": "register", | string | |
| "consumerDisplay": "register", | string | |
| "operatorCallButton": "register", | string | |
| "handsetSensor": "register", | string | |

| | |
|--|--------|
| "generalInputPort": "register", | string |
| "headsetMicrophone": "register", | string |
| "cabinetDoor": "register", | string |
| "safeDoor": "register", | string |
| "vandalShield": "register", | string |
| "cabinetFront": "register", | string |
| "cabinetRear": "register", | string |
| "cabinetRight": "register", | string |
| "cabinetLeft": "register", | string |
| "openCloseIndicator": "register", | string |
| "fasciaLight": "register", | string |
| "audioIndicator": "register", | string |
| "heatingIndicator": "register", | string |
| "consumerDisplayBacklight": "register", | string |
| "signageDisplay": "register", | string |
| "transactionIndicator": "register", | string |
| "generalOutputPort": "register", | string |
| "volumeControl": "register", | string |
| "ups": "register", | string |
| "remoteStatusMonitor": "register", | string |
| "audibleAlarm": "register", | string |
| "enhancedAudioControl": "register", | string |
| "enhancedMicrophoneControl": "register", | string |
| "microphoneVolume": "register", | string |
| "cardUnitGuidelight": "register", | string |
| "pinPadGuideLight": "register", | string |

```

"noteDispenserGuideLight": "register",      string
"coinDispenserGuideLight": "register",      string
"receiptPrinterGuideLight": "register",     string
"passbookPrinterGuideLight": "register",    string
"envelopeDepositGuideLight": "register",   string
"checkUnitGuideLight": "register",         string
"billAcceptorGuideLight": "register",      string
"envelopeDispenserGuideLight": "register", string
"documentPrinterGuideLight": "register",   string
"coinAcceptorGuideLight": "register",      string
"scannerGuideLight": "register",           string
"additionalProperties": "register",        string
"extra": {
}
}

```

Properties

[operatorSwitch](#)

Specifies whether the Operator Switch should report whenever the switch changes the operating mode:

- register - Report when this sensor is triggered.
- deregister - Do not report when this sensor is triggered.

[tamperSensor](#)

Specifies whether the Tamper Sensor should report whenever someone tampers with the terminal. See [operatorSwitch](#) for the possible values.

[intTamperSensor](#)

Specifies whether the Internal Tamper Sensor should report whenever someone tampers with the internal alarm. See [operatorSwitch](#) for the possible values.

seismicSensor

Specifies whether the Seismic Sensor should report whenever any seismic activity is detected. See [operatorSwitch](#) for the possible values.

heatSensor

Specifies whether the Heat Sensor should report whenever any excessive heat is detected. See [operatorSwitch](#) for the possible values.

proximitySensor

Specifies whether the Proximity Sensor should report whenever any movement is detected close to the terminal. See [operatorSwitch](#) for the possible values.

ambientLightSensor

Specifies whether the Ambient Light Sensor should report whenever it detects changes in the ambient light. See [operatorSwitch](#) for the possible values.

enhancedAudio

Specifies whether the Audio Jack should report whenever it detects changes in the audio jack. See [operatorSwitch](#) for the possible values.

bootSwitch

Specifies whether the Boot Switch should report whenever the delayed effect boot switch is used. See [operatorSwitch](#) for the possible values.

consumerDisplay

Specifies whether the Consumer Display Sensor should report whenever it detects changes to the consumer display. See [operatorSwitch](#) for the possible values.

operatorCallButton

Specifies whether the Operator Call Button should report whenever the Operator Call Button is pressed or released. See [operatorSwitch](#) for the possible values.

handsetSensor

Specifies whether the Handset Sensor should report whenever it detects changes of its status. See [operatorSwitch](#) for the possible values.

generalInputPort

Specifies whether the General-Purpose Input Port should report whenever it detects changes to any one of the General-Purpose Input Ports. See [operatorSwitch](#) for the possible values.

headsetMicrophone

Specifies whether the Microphone Jack should report whenever it detects changes in the microphone jack. See [operatorSwitch](#) for the possible values.

cabinetDoor

Specifies whether the Cabinet Doors should report whenever the doors are opened, closed, bolted or locked. See [operatorSwitch](#) for the possible values.

safeDoor

Specifies whether the Safe Doors should report whenever the doors are opened, closed, bolted or locked. See [operatorSwitch](#) for the possible values.

vandalShield

Specifies whether the Vandal Shield should report whenever the shield changed position. See [operatorSwitch](#) for the possible values.

cabinetFront

Specifies whether the front Cabinet Doors should report whenever the front doors are opened, closed, bolted or locked. See [operatorSwitch](#) for the possible values.

cabinetRear

Specifies whether the rear Cabinet Doors should report whenever the front doors are opened, closed, bolted or locked. See [operatorSwitch](#) for the possible values.

cabinetRight

Specifies whether the right Cabinet Doors should report whenever the front doors are opened, closed, bolted or locked. See [operatorSwitch](#) for the possible values.

cabinetLeft

Specifies whether the left Cabinet Doors should report whenever the front doors are opened, closed, bolted or locked. See [operatorSwitch](#) for the possible values.

openCloseIndicator

Specifies whether the Open/Closed Indicator should report whenever it is turned on (set to open) or turned off (set to closed). See [operatorSwitch](#) for the possible values.

fasciaLight

Specifies whether the Fascia Light should report whenever it is turned on or turned off. See [operatorSwitch](#) for the possible values.

audioIndicator

Specifies whether the Audio Indicator should report whenever it is turned on or turned off. See [operatorSwitch](#) for the possible values.

heatingIndicator

Specifies whether the Heating device should report whenever it is turned on or turned off. See [operatorSwitch](#) for the possible values.

consumerDisplayBacklight

Specifies whether the Consumer Display Backlight should report whenever it is turned on or turned off. See [operatorSwitch](#) for the possible values.

signageDisplay

Specifies whether the Signage Display should report whenever it is turned on or turned off. See [operatorSwitch](#) for the possible values.

transactionIndicator

Specifies whether the Transaction Indicators should report whenever any one of them is turned on or turned off. See [operatorSwitch](#) for the possible values.

generalOutputPort

Specifies whether the General-Purpose Output Ports should report whenever any one of them is turned on or turned off. See [operatorSwitch](#) for the possible values.

volumeControl

Specifies whether the Volume Control device should report whenever it is changed. See [operatorSwitch](#) for the possible values.

ups

Specifies whether the UPS device should report whenever it is changed. See [operatorSwitch](#) for the possible values.

remoteStatusMonitor

Specifies whether the Remote Status Monitor device should report whenever it is changed. See [operatorSwitch](#) for the possible values.

audibleAlarm

Specifies whether the Audible Alarm device should report whenever it is changed. See [operatorSwitch](#) for the possible values.

enhancedAudioControl

Specifies whether the Enhanced Audio Controller should report whenever it changes status (assuming the device is capable of generating events). See [operatorSwitch](#) for the possible values.

enhancedMicrophoneControl

Specifies whether the Enhanced Microphone Controller should report whenever it changes status (assuming the device is capable of generating events). See [operatorSwitch](#) for the possible values.

microphoneVolume

Specifies whether the Microphone Volume Control device should report whenever it is changed. See [operatorSwitch](#) for the possible values.

cardUnitGuidelight

Specifies whether the Guidance Light Indicator on the Card Unit (IDC) should report whenever it changes status. See [operatorSwitch](#) for the possible values.

pinPadGuideLight

Specifies whether the Guidance Light Indicator on the PIN pad unit should report whenever it changes status. See [operatorSwitch](#) for the possible values.

noteDispenserGuideLight

Specifies whether the Guidance Light Indicator on the notes dispenser unit should report whenever it changes status. See [operatorSwitch](#) for the possible values.

coinDispenserGuideLight

Specifies whether the Guidance Light Indicator on the coin dispenser unit should report whenever it changes status. See [operatorSwitch](#) for the possible values.

receiptPrinterGuideLight

Specifies whether the Guidance Light Indicator on the receipt printer unit should report whenever it changes status. See [operatorSwitch](#) for the possible values.

passbookPrinterGuideLight

Specifies whether the Guidance Light Indicator on the passbook printer unit should report

whenever it changes status. See [operatorSwitch](#) for the possible values.

envelopeDepositGuideLight

Specifies whether the Guidance Light Indicator on the envelope deposit unit should report whenever it changes status. See [operatorSwitch](#) for the possible values.

checkUnitGuideLight

Specifies whether the Guidance Light Indicator on the check processing unit should report whenever it changes status. See [operatorSwitch](#) for the possible values.

billAcceptorGuideLight

Specifies whether the Guidance Light Indicator on the bill acceptor unit should report whenever it changes status. See [operatorSwitch](#) for the possible values.

envelopeDispenserGuideLight

Specifies whether the Guidance Light Indicator on the envelope dispenser unit should report whenever it changes status. See [operatorSwitch](#) for the possible values.

documentPrinterGuideLight

Specifies whether the Guidance Light Indicator on the document printer unit should report whenever it changes status. See [operatorSwitch](#) for the possible values.

coinAcceptorGuideLight

Specifies whether the Guidance Light Indicator on the coin acceptor unit should report whenever it changes status. See [operatorSwitch](#) for the possible values.

scannerGuideLight

Specifies whether the Guidance Light Indicator on the scanner unit should report whenever it changes status. See [operatorSwitch](#) for the possible values.

additionalProperties

Specifies whether the vendor dependent sensors should report whenever they change

status. See `operatorSwitch` for the possible values.

extra

Specifies a vendor-specific object.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "invalidPort" | string | |
| } | | |

Properties

`completionCode`

success if the command was successful otherwise error

`errorDescription`

If not success, then this is optional vendor dependent information to provide additional information

`errorCode`

Specifies the error code if applicable. The following values are possible:

"invalidPort": An attempt to register for or disable events to a port was invalid because the port does not exist. "syntaxError": The command was invoked with incorrect input data. E.g. an attempt to both register and disable events to the same port was made.

Event Messages

- SensorsAndIndicators.PortErrorEvent

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

13.2.5 - SensorsAndIndicators.SetPorts

This command is used to set or clear one or more output ports (indicators) in the Sensors and Indicators Unit.

Command Message

| Payload | Type | Required |
|-----------------------------------|-----------------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "cabinetDoor": "noChange", | string | |
| "safeDoor": "noChange", | string | |
| "vandalShield": "noChange", | string | |
| "frontCabinetDoor": "noChange", | string | |
| "rearCabinetDoor": "noChange", | string | |
| "leftCabinetDoor": "noChange", | string | |
| "rightCabinetDoor": "noChange", | string | |
| "openClose": "noChange", | string | |
| "fasciaLight": "noChange", | string | |
| "audio": "noChange", | string | |
| "audioSound": "keypress", | string | |
| "audioContinuous": "continuous", | string | |
| "heating": "noChange", | string | |
| "displayBackLight": "noChange", | string | |
| "signageDisplay": "noChange", | string | |
| "transactionIndicators": [false], | array (boolean) | |
| "generalOutputPort": [false], | array (boolean) | |
| "volume": 0, | integer | |
| "UPS": "noChange", | string | |

```

"remoteStatusMonitor": { object
  "greenLED": "noChange", string
  "amberLED": "noChange", string
  "redLED": "noChange" string
},
"audibleAlarm": "noChange", string
"enhancedAudioControl": "noChange", string
"enhancedMicrophoneControl": "notAvailable", string
"microphoneVolume": 0, integer
"guideLight": { object
  "flashRate": "off", string
  "color": "default", string
  "direction": "entry" string
}
}

```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

cabinetDoor

Specifies all cabinet doors.

safeDoor

Specifies whether the Vandal Shield should change position.

vandalShield

Specifies whether the Vandal Shield should change position.

frontCabinetDoor

Specifies whether the front Cabinet Doors should be bolted or unbolted.

rearCabinetDoor

Specifies whether the rear Cabinet Doors should be bolted or unbolted..

leftCabinetDoor

Specifies whether the Left Cabinet Doors should be bolted or unbolted..

rightCabinetDoor

Specifies whether the right Cabinet Doors should be bolted or unbolted.s.

openClose

Specifies whether the Open/Closed Indicator should show Open or Close to a consumer.

fasciaLight

Specifies whether the Fascia Lights should be turned on or off.

audio

Specifies whether the Audio Indicator should be turned on or off.

audioSound

Specifies the Audio sound.

audioContinuous

Specifies whether the Audio Indicator sound is continuous.

heating

Specifies whether the Internal Heating device should be turned on or off.

displayBackLight

Specifies whether the Consumer Display Backlight should be turned on or off.

signageDisplay

Specifies whether the Consumer Display Backlight should be turned on or off.

transactionIndicators

Specifies whether the Transaction Indicators should be turned on or off. All Transaction Indicators must be specified and each array element represents one Transaction Indicator.

generalOutputPort

Specifies whether the General-Purpose Output Ports should be turned on or off. All General-Purpose Output Ports must be specified and each index of this value represents one General-Purpose Output Port.

volume

Specifies whether the value of the Volume Control should be changed. If so, the value of Volume Control is defined in an interval from 1 to 1000 where 1 is the lowest volume level and 1000 is the highest volume level.

UPS

Specifies whether the UPS device should be engaged or disengaged. The UPS device should not be engaged when the charge level is low.

remoteStatusMonitor

Specifies whether the state of the Remote Status Monitor device should be changed.

Specified as WFS_SIU_NO_CHANGE.

remoteStatusMonitor/greenLED

Specifies the state of the green LED.

remoteStatusMonitor/amberLED

Specifies the state of the amber LED.

remoteStatusMonitor/redLED

Specifies the state of the red LED.

audibleAlarm

Specifies whether the state of the Audible Alarm device should be changed.

enhancedAudioControl

Specifies whether the state of the Enhanced Audio Controller should be changed.

enhancedMicrophoneControl

Specifies whether the state of the Enhanced Microphone Controller should be changed.

microphoneVolume

Specifies whether the value of the Microphone Volume Control should be changed. If so, the value of Microphone Volume Control is defined in an interval from 1 to 1000 where 1 is the lowest volume level and 1000 is the highest volume level.

guideLight

Specifies the index of the guidance light to set as one of the values defined within the capabilities section:

guideLight/flashRate

Indicates which flash rates are supported by the guidelight.

guideLight/color

Indicates which colors are supported by the guidelight.

guideLight/direction

Indicates which directions are supported by the guidelight. and it's an optional field

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "invalidPort" | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

"invalidPort": An attempt to set a port to a new value was invalid because the port does not exist or the port is pre-configured as an input port. "syntaxError": The command was invoked with incorrect input data.

Event Messages

- SensorsAndIndicators.PortErrorEvent

13.2.6 - SensorsAndIndicators.SetDoor

This command is used to set the status of one of the doors.

Command Message

| Payload | Type | Required |
|------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "door": "cabinetDoor", | string | |
| "doorState": "bolt" | string | |
| } | | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

door

Specifies the door to set.

doorState

Specifies if the Cabinet or Safe doors should be bolted or unbolted or if the position of the Vandal Shield should be changed.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "invalidPort" | string | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

"invalidPort": An attempt to set a port to a new value was invalid because the port does not exist or the port is pre-configured as an input port. "PortError": A hardware error occurred while executing the command. "syntaxError": The command was invoked with incorrect input data.

Event Messages

- [SensorsAndIndicators.PortErrorEvent](#)

13.2.7 - SensorsAndIndicators.SetIndicator

This command is used to set the status of an indicator.

Command Message

| Payload | Type | Required |
|------------------------------|-----------------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "indicator": "openClose", | string | |
| "indicatorState": "closed", | string | |
| "generalOutputPort": [false] | array (boolean) | |
| } | | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

indicator

Specifies the indicator to set.

indicatorState

Specifies the commands for the Open/Close Indicator, Fascia Light, Audio Indicator, Heating device, Consumer Display Backlight, Signage Display and General-Purpose Output Ports.

generalOutputPort

For General-Purpose Output Ports specifies whether the General-Purpose Output Ports should be turned on or off. All General-Purpose Output Ports must be specified and each index of this array represents one General-Purpose Output Port.

Completion Message

| Payload | Type | Required |
|---------|------|----------|
|---------|------|----------|

```
{  
  "completionCode": "success",           string  
  "errorDescription": Add example to YAML, string  
  "errorCode": "invalidPort"           string  
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

"invalidPort": An attempt to set a port to a new value was invalid because the port does not exist or the port is pre-configured as an input port. "PortError": A hardware error occurred while executing the command. "syntaxError": The command was invoked with incorrect input data.

Event Messages

- SensorsAndIndicators.PortErrorEvent

13.2.8 - SensorsAndIndicators.SetAutostartupTime

This command is used to set the time at which the machine will automatically start. It is also used to disable automatic start-up.

Command Message

| Payload | Type | Required |
|-------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "wMode": "clear", | string | |
| "startTime": { | object | |
| "year": 0, | integer | |
| "month": 0, | integer | |
| "dayOfWeek": 0, | integer | |
| "day": 0, | integer | |
| "hour": 0, | integer | |
| "minute": 0 | integer | |
| } | | |
| } | | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

wMode

Specifies the current auto start-up control mode configured.

startTime

Specifies the current auto start-up time configuration.

startTime/year

Specifies the year. The value should be ignored if it is not relevant to the mode value..

startTime/month

Specifies the month. The value should be ignored if it is not relevant to the mode value..

startTime/dayOfWeek

Specifies the day of the week, in values from 0 (Sunday) to 6 (Saturday). The value should be ignored if it is not relevant to the mode value.

startTime/day

Specifies the day. The value should be ignored if it is not relevant to the mode value..

startTime/hour

Specifies the hour. The value should be ignored if it is not relevant to the mode value..

startTime/minute

Specifies the minute. The value should be ignored if it is not relevant to the mode value..

Completion Message

| Payload | Type | Required |
|---|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

Event Messages

None

13.3 - Unsolicited Messages

13.3.1 - SensorsAndIndicators.PortErrorEvent

This event is used to specify that a port has detected an error.

| Payload | Type | Required |
|---|--------|----------|
| { | | |
| "portType": "operatorSwitch", | string | |
| "portError": "invalidPort", | string | |
| "portStatus": { | object | |
| "operatorSwitchState": "notAvailable", | string | |
| "tamperSensorState": "notAvailable", | string | |
| "intTamperSensorState": "notAvailable", | string | |
| "seismicSensorState": "notAvailable", | string | |
| "heatSensorState": "notAvailable", | string | |
| "proximitySensorState": "notAvailable", | string | |
| "ambientLightSensorState": "notAvailable", | string | |
| "enhancedAudioSensorState": "notAvailable", | string | |
| "bootSwitchSensorState": "notAvailable", | string | |
| "displaySensorState": "notAvailable", | string | |

| | |
|--|---------|
| "operatorCallButtonSensorState": "notAvailable", | string |
| "handsetSensorState": "notAvailable", | string |
| "generalInputPortNumber": 0, | integer |
| "generalInputPortState": "on", | string |
| "headsetMicrophoneSensorState": "notAvailable", | string |
| "fasciaMicrophoneSensorState": "notAvailable", | string |
| "cabinetDoorState": "notAvailable", | string |
| "safeDoorState": "notAvailable", | string |
| "vandalShieldState": "notAvailable", | string |
| "cabinetFrontDoorState": "notAvailable", | string |
| "cabinetRearDoorState": "notAvailable", | string |
| "cabinetLeftDoorState": "notAvailable", | string |
| "cabinetRightDoorState": "notAvailable", | string |
| "openClosedIndicatorState": "notAvailable", | string |
| "fasciaLightState": "notAvailable", | string |
| "audioState": "notAvailable", | string |
| "heating": "notAvailable", | string |
| "consumerDisplayBacklight": "notAvailable", | string |
| "signageDisplay": "notAvailable", | string |
| "transactionIndicator": [| array |
| "State": 0, | integer |
| "lampNumber": 0 | integer |
| "generalOutputPort": [| array |
| "State": 0, | integer |
| "generalOutputPortNumber": 0 | integer |
| "availability": "available", | string |

```

"volume": 0,                                integer
"volumeLevel": 0                            integer
"UPSSState": "notAvailable",                string
"remoteStatusMonitorState": "greenLEDOn",    string
"audibleAlarm": "notAvailable",              string
"enhancedAudioControlState": "notAvailable", string
"enhancedMicrophoneControlState": "notAvailable", string
"flashRate": "notAvailable",                 string
"colour": "red",                            string
"direction": "entry",                      string
"position": "default"                      string
},
"extra": [Add example to YAML]           array (string)
}

```

Properties

portType

Specifies the sensor or indicator that has detected an error.

portError

Specifies the error of the port.

portStatus

Specifies the state of an individual port.

portStatus/tamperSensorState

Specifies the state of the Tamper sensor.

portStatus/intTamperSensorState

Specifies the state of the internal Tamper sensor.

portStatus/seismicSensorState

Specifies the state of the Seismic sensor.

portStatus/heatSensorState

Specifies the state of the heat sensor.

portStatus/proximitySensorState

Specifies the state of the proximity sensor.

portStatus/ambientLightSensorState

Specifies the state of the ambient light sensor.

portStatus/enhancedAudioSensorState

Specifies the state of the enhanced audio sensor.

portStatus/bootSwitchSensorState

Specifies the state of the boot switch sensor.

portStatus/displaySensorState

Specifies the state of the Consumer Display.

portStatus/operatorCallButtonSensorState

Specifies the state of the operator call button sensor.

portStatus/handsetSensorState

Specifies the state of the Handset.

portStatus/generalInputPortNumber

Specifies the number of the general Purpose Input Port.

portStatus/generalInputPortState

Specifies the state of the general Purpose Input Port.

portStatus/headsetMicrophoneSensorState

Specifies the state of the headset microphone sensor.

portStatus/fasciaMicrophoneSensorState

Specifies the state of the fascia microphone sensor.

portStatus/cabinetDoorState

Specifies the state of the cabinet door.

portStatus/safeDoorState

Specifies the state of the safe door.

portStatus/vandalShieldState

Specifies the state of the vandal shield door.

portStatus/cabinetFrontDoorState

Specifies the state of the front cabinet door.

portStatus/cabinetRearDoorState

Specifies the state of the rear cabinet door.

portStatus/cabinetLeftDoorState

Specifies the state of the left cabinet door.

portStatus/cabinetRightDoorState

Specifies the state of the right cabinet door.

portStatus/openClosedIndicatorState

Specifies the Open/Closed state of the indicator.

portStatus/fasciaLightState

Specifies the fascia light state.

portStatus/audioState

Specifies the audio state of the indicator.

portStatus/heating

Specifies the internal heating state.

portStatus/consumerDisplayBacklight

Specifies the consumer display backlight state.

portStatus/signageDisplay

Specifies the signage display state.

portStatus/transactionIndicator

Specifies the transaction indicator state.

portStatus/generalOutputPort

Specifies the state of the vendor dependent General-Purpose Output Ports.

portStatus/availability

Specifies whether the microphone volume control status is available.

portStatus/volume

Specifies the value of the microphone volume control.

portStatus/UPSState

Specifies the state of the UPS.

portStatus/remoteStatusMonitorState

Specifies the state of the remote Status Monitor.

portStatus/audibleAlarm

Specifies the audible alarm state.

portStatus/enhancedAudioControlState

Specifies the state of the Enhanced Audio Controller.

portStatus/enhancedMicrophoneControlState

Specifies the state of the Enhanced Microphone Controller.

portStatus/flashRate

Indicates the guidelight flash rate. The following values are possible: "notAvailable": The light indicator is not available. "off": The light is turned off. "slow": The light is blinking slowly. "medium": The light is blinking medium frequency. "quick": The light is blinking quickly. "continuous": The light is continuous (steady).

portStatus/colour

Indicates the guidelight colour. The following values are possible: "defaultColor": The light indicator is not available. "red": The light is red. "green": The light is green. "yellow": The

light us yellow. "blue": The light is blue. "cyan": The light is cyan. "magenta": The light is magenta. "white": The light is white.

portStatus/direction

Indicates the guidelight direction. The following values are possible: "entry": The light is indicating entry. "exit": The light is indicating exit.

portStatus/position

Indicates the guidelight position. The following values are possible: "default": The default position. "left": The left position. "right": The right position. "center": The center position. "top": The top position. "bottom": The bottom position. "front": The front position. "rear": The rear position.

extra

Specifies a list of vendor-specific, or any other extended, information. The information is returned as a series of "key=value" strings so that it is easily extendable by Service Providers.

14 - Barcode Reader Interface

This chapter defines the Barcode Reader interface functionality and messages.

14.1 - Summary

A Barcode Reader scans barcodes using any scanning technology. The device logic converts light signals or image recognition into client data and transmits it to the host system.

14.2 - Command Messages

14.2.1 - `BarcodeReader.Read`

This command enables the barcode reader. The barcode reader will scan for barcodes and when it successfully manages to read one or more barcodes the command will complete. The completion event for this command contains the scanned barcode data.

The device waits for the period of time specified by the `timeout` parameter for one of the enabled symbologies to be presented, unless the hardware has a fixed timeout period that is less than the value passed in the command.

Command Message

| Payload | Type | Required |
|----------------------------------|---------|----------|
| { | | |
| <code>"timeout": 5000,</code> | integer | |
| <code>"symbologies": {</code> | object | |
| <code> "ean128": false,</code> | boolean | |
| <code> "ean8": false,</code> | boolean | |
| <code> "ean8_2": false,</code> | boolean | |
| <code> "ean8_5": false,</code> | boolean | |
| <code> "ean13": false,</code> | boolean | |
| <code> "ean13_2": false,</code> | boolean | |

| | |
|-----------------------|---------|
| "ean13_5": false, | boolean |
| "jan13": false, | boolean |
| "upcA": false, | boolean |
| "upcE0": false, | boolean |
| "upcE0_2": false, | boolean |
| "upcE0_5": false, | boolean |
| "upcE1": false, | boolean |
| "upcE1_2": false, | boolean |
| "upcE1_5": false, | boolean |
| "upcA_2": false, | boolean |
| "upcA_5": false, | boolean |
| "codabar": false, | boolean |
| "itf": false, | boolean |
| "code11": false, | boolean |
| "code39": false, | boolean |
| "code49": false, | boolean |
| "code93": false, | boolean |
| "code128": false, | boolean |
| "msi": false, | boolean |
| "plessey": false, | boolean |
| "std20f5": false, | boolean |
| "std20f5Iata": false, | boolean |
| "pdf417": false, | boolean |
| "microPdf417": false, | boolean |
| "dataMatrix": false, | boolean |
| "maxiCode": false, | boolean |

| | |
|---------------------------|---------|
| "codeOne": false, | boolean |
| "channelCode": false, | boolean |
| "telepenOriginal": false, | boolean |
| "telepenAim": false, | boolean |
| "rss": false, | boolean |
| "rssExpanded": false, | boolean |
| "rssRestricted": false, | boolean |
| "compositeCodeA": false, | boolean |
| "compositeCodeB": false, | boolean |
| "compositeCodeC": false, | boolean |
| "posiCodeA": false, | boolean |
| "posiCodeB": false, | boolean |
| "triopticCode39": false, | boolean |
| "codablockF": false, | boolean |
| "code16K": false, | boolean |
| "qrCode": false, | boolean |
| "aztec": false, | boolean |
| "ukPost": false, | boolean |
| "planet": false, | boolean |
| "postnet": false, | boolean |
| "canadianPost": false, | boolean |
| "netherlandsPost": false, | boolean |
| "australianPost": false, | boolean |
| "japanesePost": false, | boolean |
| "chinesePost": false, | boolean |
| "koreanPost": false | boolean |

}

}

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

symbologies

Specifies the sub-set of bar code symbologies that the client wants to be accepted for this command. In some cases the Service Provider can discriminate between barcode symbologies and return the data only if the presented symbology matches with one of the desired symbologies. See the [canFilterSymbologies](#) capability to determine if the Service Provider supports this feature. If the Service Provider does not support this feature then this parameter is ignored. If all symbologies should be accepted then the *symbologies* field should be omitted.

symbologies/ean128

GS1-128

symbologies/ean8

EAN-8

symbologies/ean8_2

EAN-8 with 2 digit add-on

symbologies/ean8_5

EAN-8 with 5 digit add-on

symbologies/ean13

EAN13

symbologies/ean13_2

EAN-13 with 2 digit add-on

symbologies/ean13_5

EAN-13 with 5 digit add-on

symbologies/jan13

jan-13

symbologies/upcA

UPC-A

symbologies/upcE0

UPC-E

symbologies/upcE0_2

UPC-E with 2 digit add-on

symbologies/upcE0_5

UPC-E with 5 digit add-on

symbologies/upcE1

UPC-E with leading 1

symbologies/upcE1_2

UPC-E with leading 1and 2 digit add-on

symbologies/upcE1_5

UPC-E with leading 1 and 5 digit add-on

symbologies/upcA_2

UPC-A with 2 digit add-on

symbologies/upcA_5

UPC-A with 5 digit add-on

symbologies/codabar

CODABAR (NW-7)

symbologies/itf

Interleaved 2 of 5 (ITF)

symbologies/code11

CODE 11 (USD-8)

symbologies/code39

CODE 39

symbologies/code49

CODE 49

symbologies/code93

CODE 93

symbologies/code128

CODE 128

symbologies/msi

MSI

symbologies/plessey

PLESSEY

symbologies/std2Of5

STANDARD 2 of 5 (INDUSTRIAL 2 of 5 also)

symbologies/std2Of5Iata

STANDARD 2 of 5 (IATA Version)

symbologies/pdf417

PDF-417

symbologies/microPdf417

MICROPDF-417

symbologies/dataMatrix

GS1 DataMatrix

symbologies/maxiCode

MAXICODE

symbologies/codeOne

CODE ONE

symbologies/channelCode

CHANNEL CODE

symbologies/telepenOriginal

Original TELEPEN

symbologies/telepenAim

AIM version of TELEPEN

symbologies/rss

GS1 DataBar™

symbologies/rssExpanded

Expanded GS1 DataBar™

symbologies/rssRestricted

Restricted GS1 DataBar™

symbologies/compositeCodeA

Composite Code A Component

symbologies/compositeCodeB

Composite Code B Component

symbologies/compositeCodeC

Composite Code C Component

symbologies/posiCodeA

Posicode Variation A

symbologies/posiCodeB

Posicode Variation B

symbologies/triopticCode39

Trioptic Code 39

symbologies/codablockF

Codablock F

symbologies/code16K

Code 16K

symbologies/qrCode

QR Code

symbologies/aztec

Aztec Codes

symbologies/ukPost

UK Post

symbologies/planet

US Postal Planet

symbologies/postnet

US Postal Postnet

symbologies/canadianPost

Canadian Post

symbologies/netherlandsPost

Netherlands Post

symbologies/australianPost

Australian Post

symbologies/japanesePost

Japanese Post

symbologies/chinesePost

Chinese Post

symbologies/koreanPost

Korean Post

Completion Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "barcodeInvalid", | string | |
| "readOutput": [{ | array (object) | |
| "symbology": Add example to YAML, | string | |
| "barcodeData": Add example to YAML, | string | |
| "symbologyName": Add example to YAML | string | |
| } | | |

}

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- barcodeInvalid - The read operation could not be completed successfully. The barcode presented was defective or was wrongly read.

readOutput

An array of barcode data structures

readOutput/symbology

Specifies the barcode symbology recognized. This contains one of the values returned in the [symbologies](#) field of the [Common.Capabilities](#) command. If the barcode reader is unable to recognize the symbology as one of the values reported via the device capabilities then the value for this field will be *symbologyUnknown*.

readOutput/barcodeData

Contains the Base64 encoded barcode data read from the barcode reader. The format of the data will depend on the barcode symbology read. In most cases this will be an array of bytes containing ASCII numeric digits. However, the format of the data in this field depends entirely on the symbology read, e.g. it may contain 8 bit character values where the symbol is dependent on the codepage used to encode the barcode, may contain UNICODE data, or may be a binary block of data. The client is responsible for checking the completeness and validity of the data.

readOutput/symbologyName

A vendor dependent symbology identifier for the symbology recognized.

Event Messages

None

14.2.2 - BarcodeReader.Reset

This command is used to reset the device. The scanner returns to power-on initial status and remains disabled for any barcode label reading.

Command Message

| <u>Payload</u> | <u>Type</u> | <u>Required</u> |
|---------------------------|-------------|-----------------|
| { "timeout": 5000 } | integer | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| <u>Payload</u> | <u>Type</u> | <u>Required</u> |
|---|------------------|-----------------|
| { | | |
| "completionCode": "success", "errorDescription": Add example to YAML | string string | |

}

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

Event Messages

None

15 - Card Embosser Interface

This chapter defines the Card Embosser interface functionality and messages.

15.1 - Summary

Embossing card units are generally viewed by XFS as compound devices with the following capabilities and features:

- Embossing or printing of magnetic stripe card/ smart card.
- Reading/encoding magnetic stripe tracks 1, 2, and 3.
- Reading/writing smart card.
- Contactless chip card readers
- LCD display/ keypad input.

The XFS services supporting the various embossing card unit components are outlined as follows:

- Embossing or printing of magnetic stripe card/ smart card - [CardEmbosser](#) service.
- Reading / encoding magnetic stripe tracks 1, 2, and 3 - [CardReader](#) service, however when combined encoding / embossing is performed the CardEmbosser service class is used.
- Reading / writing smart cards - CardReader service, however when combined writing smart card / embossing is performed the CardEmbosser service class is used.
- LCD display / keypad input - [TextTerminal](#) service.

15.2 - General Information

15.2.1 - Embossing Form, Field and Media Definitions

This section outlines the format of the embossing definitions of forms and the fields within them.

15.2.1.1 - Definition Syntax

The syntactic rules for form, field and media definitions are as follows:

- **White space**
space, tab

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

- **Line continuation**
backslash (\)
- **Line termination**
CR, LF, CR/LF; line termination ends a “keyword section” (a keyword and its value[s])
- **Keywords**
must be all upper case
- **Names**
(field/media/font names) any case; case is preserved; Service Providers are case sensitive
- **Strings**
all strings must be enclosed in double quote characters ("); to include a double quote in a string, “escape” with a forward slash (/")
- **Comments**
start with two forward slashes (//), end at line termination

Other Notes:

- If a keyword is present, all its values must be specified; default values are used only if the keyword is absent.
- Values that are character strings are marked with asterisks in the definitions below, and must be quoted as specified above.
- The order of attributes within the forms is not mandatory and the attributes may be defined in any order.
- All forms can be represented using either ISO 646 (ANSI) or UNICODE character encoding. If the UNICODE representation is used then all Names and Strings are restricted to an internal representation of ISO 646 (ANSI) characters. Only the INITIALVALUE and FORMAT keyword values can have double byte values outside of the ISO 646 (ANSI) character set.
- If forms character encoding is UNICODE then, consistent with the UNICODE standard, the file prefix must be in little endian (xFFFE) or big endian (xFEFF) notation, such that UNICODE encoding is recognized.

15.2.1.2 - Embossing Form and Media Measurements

The UNIT keyword sections of the form and media definitions specify the base horizontal and vertical resolution as follows:

- The *base* value specifies the base unit of measurement.
- The *x* and *y* values specify the horizontal and vertical resolution as fractions of the base value (e.g. an *x* value of 10 and a base value of MM means that the base horizontal resolution is 0.1mm).

The base resolutions thus defined by the UNIT keyword section of the *form* definition are used as the units of the form definition keyword sections:

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

- SIZE (*width* and *height* values)
- ALIGNMENT (*xoffset* and *yoffset* values)

and of the field definition keyword sections:

- POSITION (*x* and *y* values)
- SIZE (*width* and *height* values)

The base resolutions thus defined by the UNIT keyword section of the *media* definition are used as the units of the media definition keyword sections:

- SIZE (*width* and *height* values)
- EMBOSSAREA (*x*, *y*, *width* and *height* values)
- RESTRICTED (*x*, *y*, *width* and *height* values)

[15.2.1.3 - Embossing Form Definition](#)

Attributes are not required in any mandatory order within a Form definition.

XFSFORM

XFSFORM *formname*

BEGIN

| | | |
|-----------------------------|--------------------|--|
| (required) UNIT | <i>base</i> , | Base resolution unit for form definition: MM INCH ROWCOLUMN |
| | <i>x</i> , | Horizontal base unit fraction |
| | <i>y</i> | Vertical base unit fraction |
| (required) SIZE | <i>width</i> , | Width of form |
| | <i>height</i> | Height of form |
| (required) ALIGNMENT | <i>alignment</i> , | Alignment of the form on the physical media TOPLEFT (default) TOPRIGHT BOTTOMLEFT BOTTOMRIGHT |
| | <i>xoffset</i> , | Horizontal offset relative to the horizontal alignment specified by alignment. Always specified as a positive value (i.e. if aligned to the right side of the media, means offset the form to the left). (default = 0) |

| | | |
|------------------------|-------------------|---|
| | <i>yoffset</i> | Vertical offset relative to the vertical alignment specified by alignment. Always specified as a positive value (i.e. if aligned to the bottom of the media, means offset the form upward). (default = 0) |
| VERSION | <i>major</i> , | Major version number |
| | <i>minor</i> , | Minor version number |
| | <i>date*</i> , | Creation/modification date |
| | <i>author*</i> | Author of form |
| COPYRIGHT | <i>copyright*</i> | Copyright entry |
| TITLE | <i>title*</i> | Title of form |
| COMMENT | <i>comment*</i> | Comment section |
| USERPROMPT | <i>prompt*</i> | Prompt string for user interaction |
| [XFSFIELD BEGIN | <i>fieldname*</i> | One field definition (as defined in the next section) for each field in the form. |
| ... | | |
| END] | | |
| END | | |

15.2.1.4 - Embossing Field Definition

| XFSFIELD | | |
|----------------------------|------------|--|
| XFSFIELD | | <i>fieldname</i> |
| BEGIN | | |
| (required) POSITION | <i>x</i> , | Horizontal position (relative to left or right side of form, depending upon HPOSITION keyword) |
| | <i>y</i> | Vertical position (relative to top or bottom of form, depending upon VPOSITION keyword) |
| HPOSITION | | Horizontal field positioning relative to: LEFT (default) RIGHT |
| VPOSITION | | Vertical field positioning relative to: TOP |

| | | |
|------------------------|--------------------------|---|
| | | BOTTOM (default) |
| SIDE | <i>side</i> | Side of card: FRONT (default) BACK |
| (required) SIZE | <i>width, height</i> | Field width Field height |
| TYPE | <i>fieldtype</i> | Type of field: TEXT (default) OCR |
| CLASS | <i>class</i> | Field class: OPTIONAL (default) STATIC REQUIRED |
| CASE | <i>case</i> | Convert field contents to: NOCHANGE (default) UPPER LOWER |
| HORIZONTAL | <i>justify</i> | Horizontal alignment of field contents: LEFT (default) RIGHT CENTER JUSTIFY |
| VERTICAL | <i>justify</i> | Vertical alignment of field contents: BOTTOM (default) CENTER TOP |
| FONT | <i>fontname*</i> | Font name; in some cases this predefines the following parameters: |
| POINTSIZE | <i>pointsize</i> | Point size. If unspecified, the point size defaults to the POINTSIZE defined for the form. |
| CPI | <i>cpi</i> | Characters per inch |
| LPI | <i>lpi</i> | Lines per inch |
| FORMAT | <i>formatstring*</i> | This is a client defined input field describing how the client should format the data. This may be interpreted by the Service Provider. |
| INITIALVALUE | <i>value*</i> | Initial value |
| END | | |

15.2.1.5 - Media Definition

The media definition determines those characteristics that result from the combination of a particular media type together with a particular vendor's identification card or smart card. The aim is to make it easy to move forms between different vendor's identification cards or smart cards which might have different constraints on how they handle a specific media type. It is the Service Provider's responsibility to ensure that the form definition does not specify the embossing of any fields that conflict with the media definition. An example of such a conflict might be that the form definition asks for a field to be embossed in an area that the media definition defines as a restricted area, such as on the chip of a smart card.

XFSMEDIA

| | | |
|------------------------|--------------------------------|---|
| XFSMEDIA | <i>medianame*</i> | |
| BEGIN | | |
| TYPE | <i>type</i> | Predefined media types are: EMBOSSCARD PRINTCARD |
| (required) UNIT | <i>base,</i> | Base resolution unit for form definition: MM INCH ROWCOLUMN |
| | <i>x,</i> | Horizontal base unit fraction |
| | <i>y</i> | Vertical base unit fraction |
| (required) SIZE | <i>width,</i> <i>height</i> | Width of physical media Height of physical media |
| EMBOSSAREA | <i>x,</i> | Embossing or Printing area relative to top left corner of physical media (default = physical size of media) |
| | <i>y,</i> | |
| | <i>width,</i> | |
| | <i>height</i> | |
| RESTRICTED | <i>x,</i> | Restricted area relative to top left corner of physical media (default = no restricted area) |
| | <i>y</i> | |
| | <i>width,</i> | |
| | <i>height</i> | |

END

15.3 - Command Messages

15.3.1 - CardEmbosser.GetFormList

This command is used to retrieve the list of forms available on the device.

Command Message

| Payload | Type | Required |
|--------------------------------------|------|----------|
| { "timeout": 5000 integer } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "formList": [Add example to YAML] | array (string) | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

formList

The list of form names.

Event Messages

None

[15.3.2 - CardEmbosser.GetQueryForm](#)

This command is used to retrieve details of the definition of a specified CardEmbosser form. GetQueryForm does not currently contain any form attributes, however it is retained for future expansion.

Command Message

| Payload | Type | Required |
|---------------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "formName": Add example to YAML | string | |
| } | | |

Properties

timeout

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

formName

The form name for which to retrieve details.

Completion Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "formNotFound", | string | |
| "formName": Add example to YAML, | string | |
| "fields": [Add example to YAML], | array (string) | |
| "charSupport": "ascii" | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

- `formNotFound` - The specified form cannot be found.
- `formInvalid` - The specified form is invalid.

formName

The form name.

fields

The field names.

charSupport

Specifies the the Character Set in which the form is encoded as one of the following:

- `ascii` - ASCII is supported for XFS forms initial data values and FORMAT strings.
- `unicode` - UNICODE is supported for XFS forms initial data values and FORMAT strings.

Event Messages

None

15.3.3 - CardEmbosser.GetMediaList

This command is used to retrieve the list of media definitions available on the device.

Command Message

| Payload | Type | Required |
|-----------------|---------|----------|
| { | | |
| "timeout": 5000 | integer | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "mediaList": [Add example to YAML] | array (string) | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

mediaList

The list of media names.

Event Messages

None

[15.3.4 - CardEmbosser.GetQueryMedia](#)

This command is used to retrieve details of the definition of a specified media.

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Command Message

| Payload | Type | Required |
|----------------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "mediaName": Add example to YAML | string | |
| } | | |

Properties**timeout**

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

mediaName

The media name for which to retrieve details.

Completion Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "mediaNotFound", | string | |
| "mediaType": "embossableCard", | string | |
| "base": "inch", | string | |
| "unitX": 0, | integer | |
| "unitY": 0, | integer | |
| "sizeWidth": 0, | integer | |

```

"sizeHeight": 0,           integer
"embossAreaX": 0,          integer
"embossAreaY": 0,          integer
"embossAreaWidth": 0,       integer
"embossAreaHeight": 0,      integer
"restrictedAreaX": 0,       integer
"restrictedAreaY": 0,       integer
"restrictedAreaWidth": 0,    integer
"restrictedAreaHeight": 0   integer
}

}

```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- mediaNotFound - The specified media definition cannot be found.
- mediaInvalid - The specified media definition is invalid.

mediaType

Specifies the type of media as one of the following:

- embossableCard - Embossable card media.
- printableCard - Printable card media.

base

Specifies the base unit of measurement of the form as one of the following:

- inch - The base unit is inches.
- mm - The base unit is millimeters.
- rowColumn - The base unit is rows and columns.

unitX

Specifies the horizontal resolution of the base units as a fraction of the wBase value. For example, a value of 16 applied to the base unit inch means that the base horizontal resolution is 1/16".

unitY

Specifies the vertical resolution of the base units as a fraction of the wBase value. For example, a value of 10 applied to the base unit mm means that the base vertical resolution is 0.1 mm.

sizeWidth

Specifies the width of the media in terms of the base horizontal resolution.

sizeHeight

Specifies the height of the media in terms of the base vertical resolution.

embossAreaX

Specifies the horizontal offset of the Card Emboss area relative to the top left corner of the media in terms of the base horizontal resolution.

embossAreaY

Specifies the vertical offset of the Card Emboss area relative to the top left corner of the media in terms of the base vertical resolution.

embossAreaWidth

Specifies the Card Emboss area width of the media in terms of the base horizontal

resolution.

embossAreaHeight

Specifies the Card Emboss area height of the media in terms of the base vertical resolution.

restrictedAreaX

Specifies the horizontal offset of the restricted area relative to the top left corner of the media in terms of the base horizontal resolution.

restrictedAreaY

Specifies the vertical offset of the restricted area relative to the top left corner of the media in terms of the base vertical resolution.

restrictedAreaWidth

Specifies the restricted area width of the media in terms of the base horizontal resolution.

restrictedAreaHeight

Specifies the restricted area height of the media in terms of the base vertical resolution.

Event Messages

None

[15.3.5 - CardEmbosser.GetQueryField](#)

This function is used to retrieve details on the definition of a single or all fields on a specified form.

Command Message

| Payload | Type | Required |
|---------|------|----------|
| { | | |

```

"timeout": 5000,           integer
"formName": Add example to YAML, string
"fieldName": Add example to YAML   string
}

```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

formName

The form name.

fieldName

The field name.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "formNotFound", | string | |
| "fields": { | object | |
| "type": "text", | string | |
| "class": "static", | string | |
| "initialValue": Add example to YAML, | string | |

```
"format": Add example to YAML           string  
}  
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- formNotFound - The specified form cannot be found.
- fieldNotFound - The specified field cannot be found.

fields

Details of the field(s) requested. For each object, the key is the field name.

fields/type

Specifies the type of field as one of the following:

- text - A text field.
- ocr - An Optical Character Recognition (OCR) field.

fields/class

Specifies the class of the field as one of the following:

- static - The field data cannot be set by the client.
- optional - The field data can be set by the client.
- required - The field data must be set by the client.

fields/initialValue

The initial value of the field when the field is written as output. The field names and values can contain UNICODE if supported by the service. This can be omitted if the parameter is not specified in the field definition.

fields/format

Format string as defined in the form for this field. This can be omitted if the parameter is not specified in the field definition.

Event Messages

None

15.3.6 - CardEmbosser.EmbossCard

This command is used to emboss or print an identification card by merging the supplied variable field data with the defined form and field data specified in the form. Optionally the magnetic stripe can be read and verified before being encoded, or a smart card can be updated.

The ATR of the chip must be obtained before issuing this command by issuing the [CardReader](#) class [CardReader.ReadRawData](#) command.

Clients can use UNICODE provided the Service Provider is UNICODE compliant.

Command Message

| Payload | Type | Required |
|-----------------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "formName": Add example to YAML, | string | |
| "mediaName": Add example to YAML, | string | |
| "fields": { | object | |
| }, | | |

```
"compareFormIOFormName": Add example to YAML, string
"compareFormIOTrackData": Add example to YAML, string
"formIOFormName": Add example to YAML, string
"formIOTrackData": Add example to YAML, string
"chipProtocol": "chipT0", string
"chipData": Add example to YAML string
}
```

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

formName

The form name.

mediaName

The media name.

fields

An object containing one or more key/value pairs where the key is a field name and the value is the field value. If the field is an index field, the key must be specified as *fieldname/index* where index specifies the zero-based element of the index field. The field names and values can contain UNICODE if supported by the service.

compareFormIOFormName

compareFormIOFormName and *compareFormIOTrackData* are used collectively when the contents of the magnetic stripe are being read and verified before the card is embossed or the magnetic stripe is encoded.

The name of the magnetic stripe form to be used, as defined in the [CardReader](#) service class.

compareFormIOTrackData

The data to be used in the form.

formIOFormName

formIOFormName and [formIOTrackData](#) are used collectively when the magnetic stripe is being encoded (after a successful magnetic stripe compare operation) and during the emboss operation.

The name of the form to be used, as defined in the [CardReader](#) service class.

formIOTrackData

The data to be used in the form.

chipProtocol

chipProtocol and [chipData](#) are used collectively when the smart card is being updated during the emboss operation. If *chipProtocol* is omitted then the smart card should not be updated during the emboss operation.

This can be one of the following:

- chipT0 - Use the T=0 protocol to communicate with the chip.
- chipT1 - Use the T=1 protocol to communicate with the chip.
- chipProtocolNotRequired - The Service Provider will automatically determine the protocol used to communicate with the chip.

chipData

The Base64 encoded data to be sent to the chip.

Completion Message

| Payload | Type | Required |
|------------------------------|--------|----------|
| { | | |
| "completionCode": "success", | string | |

"errorDescription": Add example to YAML, string

"errorCode": "formNotFound" string

}

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- formNotFound - The specified form definition cannot be found.
- formInvalid - The specified form definition is invalid.
- mediaNotFound - The specified media definition cannot be found.
- mediaInvalid - The specified media definition is invalid.
- noMedia - There is no card inside the device.
- mediaOverflow - The form overflowed the media.
- cardReaderFormNotFound - The specified [CardReader](#) form definition cannot be found.
- cardReaderFormInvalid - The specified CardReader form definition is invalid.
- invalidData - An error occurred while communicating with the chip.
- protocolNotSupported - The protocol used was not supported by the Service Provider.
- atrNotObtained - The ATR was not obtained by issuing the CardReader class [CardReader.ReadRawData](#) command.
- fieldSpecFailure - The syntax of the [fields](#) member is invalid.
- fieldError - An error occurred while processing a field, causing termination of the emboss request. A [CardEmbosser.FieldErrorEvent](#) event is posted with the details.
- embossFailure - A failure has occurred during Emboss processing. A [CardEmbosser.EmbossFailureEvent](#) event is posted with details.
- charSetData - The character set(s) supported by the Service Provider is inconsistent with the use of the [fields](#) member.

Event Messages

- [CardEmbosser.InputBinThresholdEvent](#)
- [CardEmbosser.OutputBinThresholdEvent](#)
- [CardEmbosser.RetainBinThresholdEvent](#)
- [CardEmbosser.EmbossFailureEvent](#)
- [CardEmbosser.FieldErrorEvent](#)
- [CardEmbosser.FieldWarningEvent](#)
- [CardEmbosser.MediaRemovedEvent](#)
- [CardEmbosser.TonerThresholdEvent](#)

[15.3.7 - CardEmbosser.Reset](#)

Sends a service reset to the Service Provider. Any media found in the device will be captured into the specified bin (depending on hardware). The [CardEmbosser.MediaDetectedEvent](#) event will indicate that media was found in the device on reset and will indicate the position and status of the media following completion of the command.

This command is used by a client control program to cause a device to reset itself to a known good condition.

Command Message

| Payload | Type | Required |
|----------------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "mediaControl": "inputBin" | string | |
| } | | |

[Properties](#)

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

mediaControl

Specifies the action that should be done if media is detected during the reset operation.

If *mediaControl* is omitted then the Service Provider will determine where to move any media found.

This can be one of the following:

- *inputBin* - Any media detected should be moved to the input bin.
- *outputBin* - Any media detected should be moved to the output bin.
- *retainBin* - Any media detected should be moved to the retain bin.

Completion Message

| Payload | Type | Required |
|---|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

Event Messages

- [CardEmbosser.OutputBinThresholdEvent](#)
- [CardEmbosser.RetainBinThresholdEvent](#)
- [CardEmbosser.MediaDetectedEvent](#)

15.3.8 - CardEmbosser.SupplyReplenish

After the supplies have been replenished, this command is used to indicate that one or more supplies have been replenished and are expected to be in a healthy state.

Hardware that cannot detect the level of a supply and reports on the supply's status using metrics (or some other means), must assume the supply has been fully replenished after this command is issued. The appropriate threshold event must be broadcast.

Hardware that can detect the level of a supply must update its status based on its sensors, generate a threshold event if appropriate, and succeed the command even if the supply has not been replenished. If it has already detected the level and reported the threshold before this command was issued, the command must succeed and no threshold event is required.

If any one of the specified supplies is not supported by a Service Provider, unsupportedData should be returned, and no replenishment actions will be taken by the Service Provider.

Command Message

| Payload | Type | Required |
|-------------------|---------|----------|
| { | | |
| "timeout": 5000, | integer | |
| "toner": false, | boolean | |
| "inputBin": false | boolean | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

toner

The toner supply was replenished.

inputBin

The input bin supply was replenished.

Completion Message

| Payload | Type | Required |
|---|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

Event Messages

- [CardEmbosser.InputBinThresholdEvent](#)
- [CardEmbosser.TonerThresholdEvent](#)

15.4 - Event Messages

[15.4.1 - CardEmbosser.EmbossFailureEvent](#)

This event is used to specify that an error has occurred during processing of a [CardEmbosser.EmbossCard](#) execute command.

| Payload | Type | Required |
|---------|------|----------|
|---------|------|----------|

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

```
{
  "failure": "stepperError"  string
}
```

Properties

failure

Specifies the type of failure as one of the following:

- stepperError - Stepper hardware error.
- topperFoilBreak - Topper foil has broken.
- cardFeedError - Card feed failure.
- magneticStripeError - Magnetic stripe read/write error.
- retainBinFull - Retain bin is full.
- outputBinFull - Output bin is full.
- coverOpen - Device cover is open.
- topperJam - Topper has jammed.
- stackerError - Stacker error either inside device or in output bin.
- systemError - Unknown system error.
- ocrError - OCR unit failure.
- embossLimitsExceeded - Embossing limits exceeded.
- communicationsFailure - Communications failure.
- dataFormatError - Communications data format error.
- bufferOverrun - Buffer overrun.
- preEncodeReadError - Pre-encode read error.
- preEncodeDataMatchError - Data has failed to compare during pre-encode data match step.
- inputBinEmpty - Input bin is empty.
- deviceBusy - Device is busy, unable to emboss card.
- tonerOutError - Toner or ink supply is empty or printing contrast with ribbon is not sufficient.
- mediaJam - The card is jammed. Operator intervention is required.

15.4.2 - CardEmbosser.FieldErrorEvent

This event specifies that a fatal error has occurred while processing a field.

| Payload | Type | Required |
|---------|------|----------|
|---------|------|----------|

```
{
  "formName": Add example to YAML, string
  "fieldName": Add example to YAML, string
  "failure": "required"           string
}
```

Properties

formName

The form name.

fieldName

The field name.

failure

Specifies the type of failure as one of the following:

- required - The specified field must be supplied by the client.
- staticOverwrite - The specified field is static and thus cannot be overwritten by the client.
- overflow - The value supplied for the specified fields is too long.
- notFound - The specified field does not exist.
- notRead - The specified field is not an input field.
- notWrite - An attempt was made to write to an input field.
- hardwareError - The specified field uses special hardware (e.g. OCR) and an error occurred.
- typeNotSupported - The form field type is not supported with device.
- charSetForm - The Service Provider does not support the character set specified in the form.

[15.4.3 - CardEmbosser.FieldWarningEvent](#)

This event is used to specify that a non-fatal error has occurred while processing a field.

| Payload | Type | Required |
|---------|------|----------|
|---------|------|----------|

```
{  
  "formName": Add example to YAML,  string  
  "fieldName": Add example to YAML,  string  
  "failure": "required"          string  
}
```

Properties

formName

The form name.

fieldName

The field name.

failure

Specifies the type of failure as one of the following:

- required - The specified field must be supplied by the client.
- staticOverwrite - The specified field is static and thus cannot be overwritten by the client.
- overflow - The value supplied for the specified fields is too long.
- notFound - The specified field does not exist.
- notRead - The specified field is not an input field.
- notWrite - An attempt was made to write to an input field.
- hardwareError - The specified field uses special hardware (e.g. OCR) and an error occurred.
- typeNotSupported - The form field type is not supported with device.
- charSetForm - The Service Provider does not support the character set specified in the form.

15.5 - Unsolicited Messages

15.5.1 - CardEmbosser.InputBinThresholdEvent

This event specifies that the status of the input bin has changed.

| Payload | Type | Required |
|---------------|--------|----------|
| { | | |
| "state": "ok" | string | |
| } | | |

Properties

state

Specifies the state of the input bin as one of the following:

- ok - The input bin is in a good state.
- low - The input bin is nearly empty.
- empty - The input bin is empty.

15.5.2 - CardEmbosser.OutputBinThresholdEvent

This event specifies that the status of the output bin has changed.

| Payload | Type | Required |
|---------------|--------|----------|
| { | | |
| "state": "ok" | string | |
| } | | |

Properties

state

Specifies the state of the output bin as one of the following:

- ok - The output bin is in a good state.
- full - The output bin is full.
- high - The output bin is nearly full.

[15.5.3 - CardEmbosser.RetainBinThresholdEvent](#)

This event specifies that the status of the retain bin has changed.

| Payload | Type | Required |
|---------------|--------|----------|
| { | | |
| "state": "ok" | string | |
| } | | |

Properties

state

Specifies the state of the retain bin as one of the following:

- ok - The retain bin is in a good state.
- full - The retain bin is full.
- high - The retain bin is nearly full.

[15.5.4 - CardEmbosser.MediaRemovedEvent](#)

This event is generated when a card is removed before completion of a write operation.

[15.5.5 - CardEmbosser.MediaDetectedEvent](#)

This event is generated when a media is detected in the device during a reset operation.

| Payload | Type | Required |
|------------------------|--------|----------|
| { | | |
| "position": "retained" | string | |
| } | | |

Properties

position

Specifies the media position after the reset operation as one of the following:

- retained - The media was successfully retained during the reset operation.
- removed - The media was removed during the reset operation.
- jammed - The media is jammed in the device.
- unknown - The media is in an unknown position.

[15.5.6 - CardEmbosser.TonerThresholdEvent](#)

This user event is used to specify that the state of the toner or ink supply or the state of the ribbon reached a threshold.

| Payload | Type | Required |
|-----------------------------------|------|----------|
| { "state": "full" string } | | |

Properties

state

Specifies the current state of the toner or ink supply or the state of the ribbon as one of the following:

- full - The toner, ink or ribbon in the printer is in a good state.
- low - The toner or ink in the printer is low or the print contrast with a ribbon is weak.
- out - The toner or ink in the printer is out or the print contrast with a ribbon is not sufficient any more.

16 - Biometric Interface

This chapter defines the Biometric interface functionality and messages.

16.1 - Summary

Biometrics refers to metrics related to human characteristics and biology. Biometrics authentication can be used as a form of identification and/or access control. This is an overview of biometrics, as well as an introduction to the terminology used in this document. It introduces to the concept of scanning a person's biometric data in raw image form (raw biometric data), then processing it into a smaller more concise form that is easier to manage (biometric template data). The first scan of a user is called **ENROLLMENT** as the user is effectively being enrolled into a scheme by recording their biometric data. Thereafter subsequent scans of the user can be compared to the original data in order to verify who they say they are (**VERIFICATION**), or alternatively used to identify them as a specific individual (**IDENTIFICATION**).

16.2 - General Information

16.2.1 - Enrollment

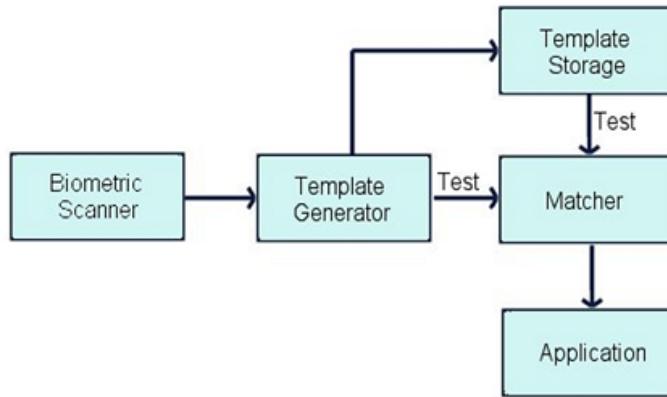
The first time an individual uses a biometric device it is called Enrollment. During enrollment, biometric data from an individual is captured and stored somewhere, for example on a smart card or in a server/host database. Normally the raw biometric data captured will be processed and converted to a smaller format that is used for subsequent comparison. This format is referred to in this document as a template. A template is a synthesis of the relevant characteristics extracted from the original raw data. Elements of the biometric data that are not used in the matching algorithm are discarded in the template to reduce the file size and to protect the identity of the enrollee.

16.2.2 - Biometric Matching

During the matching phase, the obtained template is passed to a matcher which compares it to other existing templates and a probable match is calculated, either as a Boolean true or false or as a threshold indicating the likelihood of a match. With regard to matching, biometric systems commonly have two different basic modes of operation: Verification and Identification:

Verification: performs a one-to-one comparison of captured biometric data with a specific template in order to verify that an individual is the person they claim to be.

Identification: the system performs a one-to-many comparison of captured biometric data in order to establish a person's identity.



Note: The above diagram does not make any assumptions about where the actual matching takes place. The interface provided is versatile enough to be able to support three basic Biometric systems:

Match on server: The biometric template data is stored on a server or host. When scanning takes place biometric data is sent to the server, which does the actual identification or verification.

Match on card: The biometric enrollment data for an individual is stored on a smart card/personal device. The device scans a user then returns the biometric template information to the client. This data is then sent to the card, and a client on the smart card chip does the comparison, returning the result to the client.

Match on device: The biometric enrollment data for an individual is stored on a smart card or host. The enrollment data is read from the card or host and into the device, which then compares it to scanned information, returning the result to the client.

16.2.3 - Biometric Device Types

There are many different varieties of biometric hardware, this biometrics specification supports three main different types of device:

1. **Devices which only support scanning and returning biometric data**
In this case the device is a simple biometric scanning device, User data is scanned using the [Biometric.Read](#) command, but matching is performed externally, for example on a smart card or on a server. In this case the [Biometric.Match](#) and [Biometric.SetMatch](#) commands are not supported.
2. **Devices which support a separate scan and match functionality**
These devices scan and perform a comparison as separate operations. Existing

biometric data is first imported using the [Biometric.Import](#) command. When the [Biometric.Read](#) command is then called the scanned user data is temporarily stored. The [Biometric.Match](#) command is then called to perform the comparison and return the result.

3. Devices which support a combined scan and match functionality

These devices scan and perform a comparison as a single operation. Existing biometric data is first imported using the [Biometric.Import](#) command. In this case the [Biometric.SetMatch](#) command must be called first, either as a one time call or before each [Biometric.Read](#) command. The purpose of the [Biometric.SetMatch](#) command is to set the criteria for matching. When the [Biometric.Read](#) command is then called it scans the user's biometric data and also performs the comparison as a single operation. The [Biometric.Match](#) command is then called to return the result of the comparison.

16.2.4 - Biometric Data Security

It is recommended that biometric data should be treated with the same strict caution as any other identifying and sensitive information. A well designed biometric data handling architecture should always be designed to protect against internal tampering, external attacks and other malicious threats. There are various ways of implementing good security of which three are listed below:

- **Multi Modal Biometrics**

A Uni-Modal biometric system relies on data taken from a single source of information for authentication, for example a single fingerprint reading device. In contrast, Multi-Modal biometric systems work on the premise that it is more secure to accept information from two or more biometric inputs. As an example a user could provide a fingerprint in addition to facial recognition, a positive match from two physical characteristics improves the chances of a positive identification and mitigates the possibility that biometric data has been cloned.

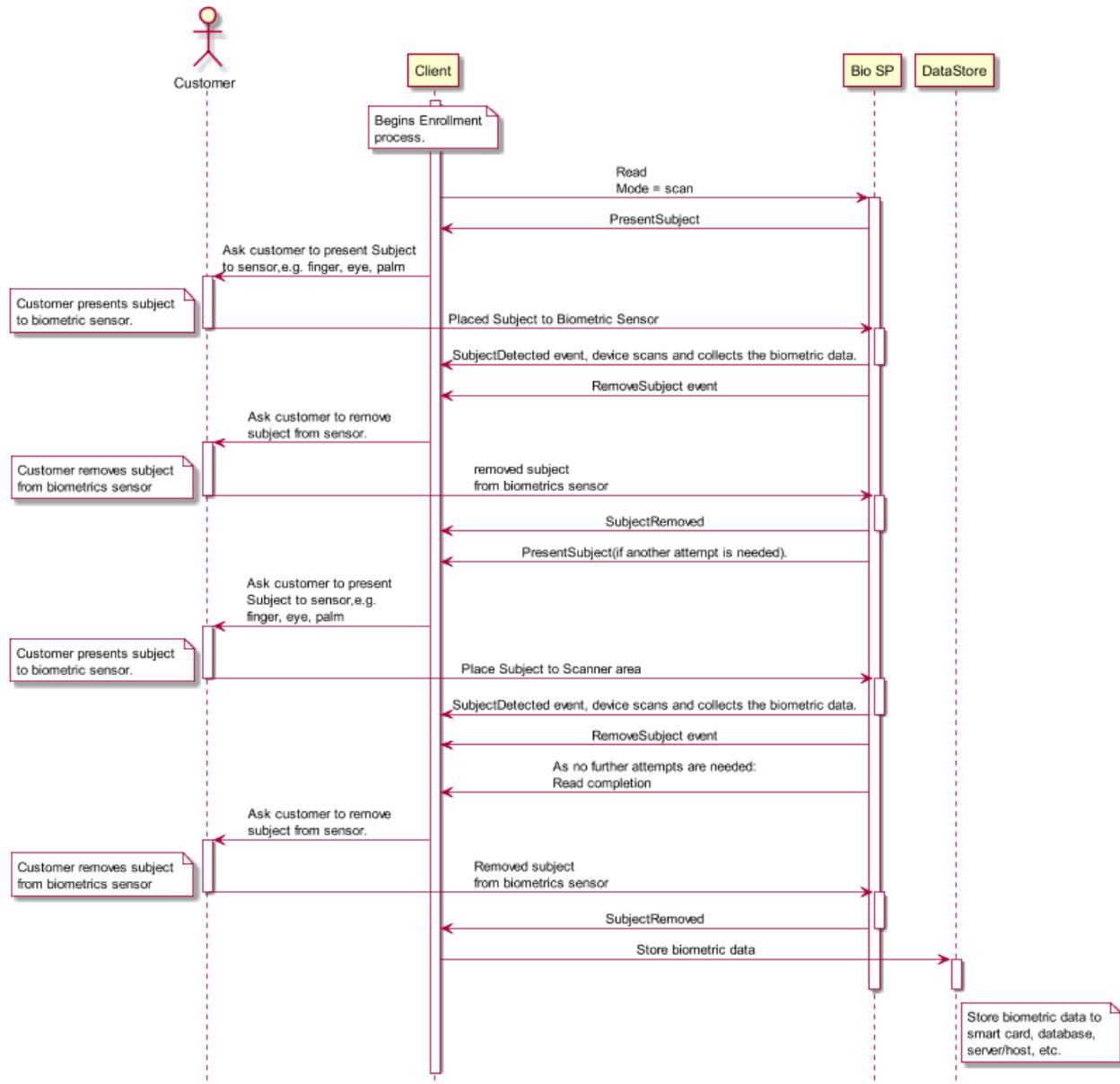
- **Data Encryption**

Biometric data should be encrypted where possible. The Biometric specification provides for this by allowing an encryption key to be specified whenever data is exchanged between a client and a Biometric Service Provider. In addition, the key management interface methods of the PIN device class can be used for key management. This can be done by using the standard compound device mechanism to implement a Biometric Service provider as a compound device together with a PIN device class Service Provider. The device compounding mechanism is described in the API specification. In this case the Biometric Service Provider would implement the biometric methods necessary to read and return data, while the key loading, reporting etc, functions of the PIN Service Provider interface would be implemented in order to provide key management.

16.2.5 - Biometric Device Command Flows

16.2.5.1 - Biometric Enrollment Command Flow

The following table describes the flow of enrolling a user using the **Biometric.Read** command. Two attempts at scanning are necessary.



16.2.5.2 - Biometric Match Command Flow – Separate Scan and Match

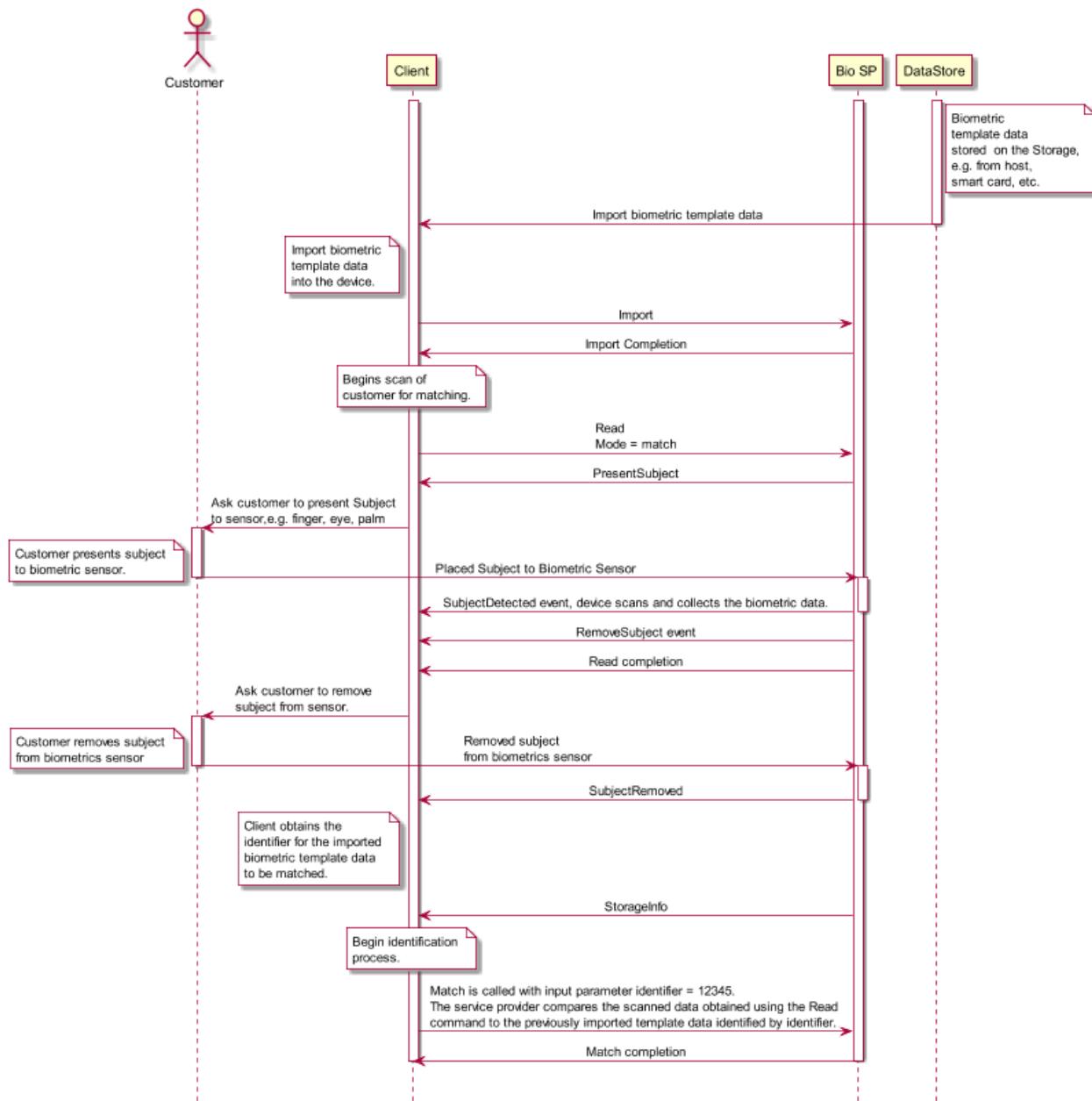
The following table describes the flow of successfully identifying a customer whose biometric template data was previously enrolled and stored on a server/smart card/host

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

system. This template data is first imported using the [Biometric.Import](#) command, which assigns it a unique identifying number. This *identifier* number can then be retrieved using the [Biometric.StorageInfo](#) command.

The [Biometric.Read](#) and [Biometric.Match](#) commands are then used to scan data and then compare it with the template identified by *identifier*. In this use case the device can perform a separate scan and match operation, therefore the [Biometric.Read](#) command is called to scan the subject's biometric data then the [Biometric.Match](#) command is called to perform the match and return the result to the client.

In this case the capability *matchSupported* is reported as `StoredMatch`.

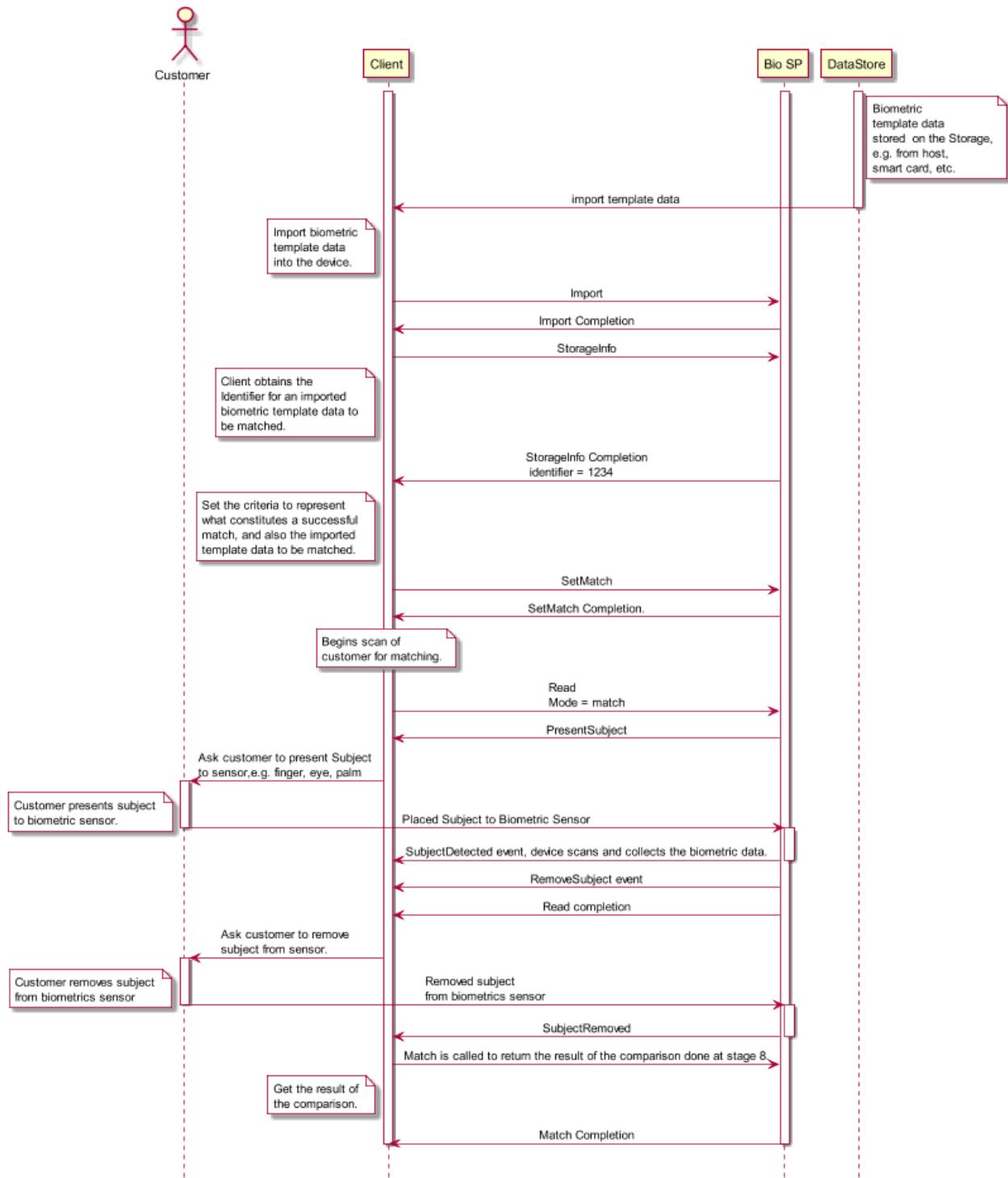


16.2.5.3 - Biometric Match Command Flow – Combined Scan and Match

The following table describes the flow of successfully identifying a customer whose biometric template data was previously enrolled and stored on a server/smart card/host system. This template data is first imported using the [Biometric.Import](#) command, which assigns it a unique identifying number. This *identifier* number can then be retrieved using the [Biometric.GetStorageInfo](#) command.

The [Biometric.Read](#), [Biometric.SetMatch](#) and [Biometric.Match](#) commands are then used to scan data and compare it with the template identified by *identifier*. In this use case the device performs a combined scan and match operation, therefore the [Biometric.SetMatch](#) command must be used to set the criteria to be used for matching, including the imported template to be identified by *identifier*. When the [Biometric.Read](#) command is then called the device scans the user and performs the comparison as a combined operation. Finally the [Biometric.Match](#) command is called to return the result of the comparison to the client.

In this case the capability *matchSupported* is reported as CombinedMatch.

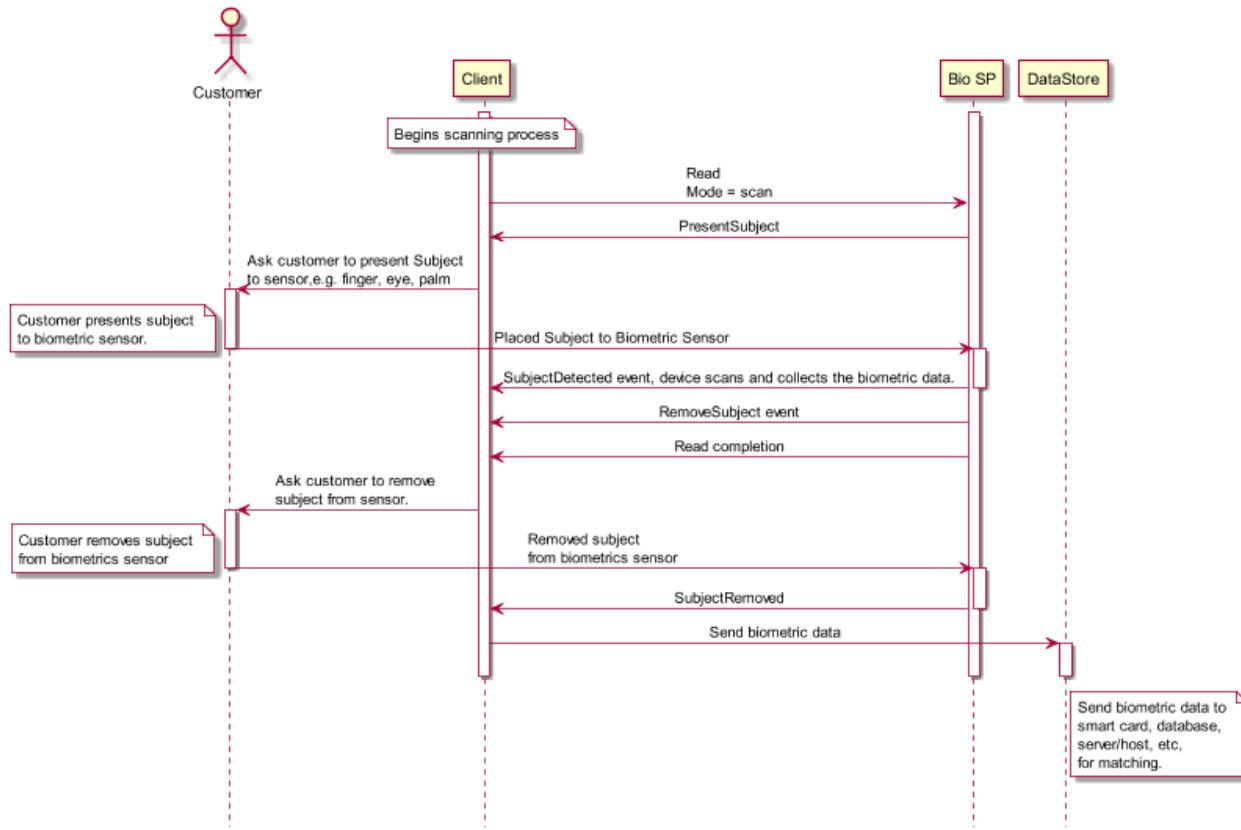


16.2.5.4 - Biometric Scan-Only Command Flow

The following table describes the flow for a simple biometric scanning device which does not support any matching at all. User data is scanned using the [Biometric.Read](#) command

All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

but matching is performed externally, for example on a smart card or on a server. In this case the capability *matchSupported* is reported as none.



16.3 - Command Messages

16.3.1 - Biometric.GetStorageInfo

This command is used to obtain information regarding the number and format of biometric templates that have been imported using the [Biometric.Import](#) command.

Command Message

| Payload | Type | Required |
|-----------------|---------|----------|
| { | | |
| "timeout": 5000 | integer | |
| } | | |

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be cancelled.

default: 0

Completion Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "noImportedData", | string | |
| "storageList": [{ | array (object) | |
| "identifier": 0, | integer | |
| "type": { | object | |
| "format": { | object | |
| "isoFid": false, | boolean | |
| "isoFmd": false, | boolean | |
| "ansiFid": false, | boolean | |
| "ansiFmd": false, | boolean | |
| "qso": false, | boolean | |
| "wso": false, | boolean | |
| "reservedRaw1": false, | boolean | |
| "reservedTemplate1": false, | boolean | |
| "reservedRaw2": false, | boolean | |
| "reservedTemplate2": false, | boolean | |
| "reservedRaw3": false, | boolean | |

```

"reservedTemplate3": false          boolean
},
"algorithm": {                      object
"ecb": false,                      boolean
"cbc": false,                      boolean
"cfb": false,                      boolean
"rsa": false                        boolean
},
"keyName": Add example to YAML     string
}
},
"$ref":                               undefined
}

```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- noImportedData - No data to return. Typically means that no data has been imported using the [Biometric.Import](#) command.

storageList

Storage Array.

storageList/identifier

A unique number which identifies the template.

storageList/type

Specifies the biometric data type of the template data.

storageList/type/format

Specifies the format of the template data.

storageList/type/format/isoFid

Raw ISO FID format

storageList/type/format/isoFmd

ISO FMD template format

storageList/type/format/ansiFid

Raw ANSI FID format

storageList/type/format/ansiFmd

ANSI FMD template format

storageList/type/format/qso

Raw QSO image format

storageList/type/format/wso

WSQ image format

storageList/type/format/reservedRaw1

Reserved for a vendor-defined Raw format.

storageList/type/format/reservedTemplate1

Reserved for a vendor-defined Template format.

storageList/type/format/reservedRaw2

Reserved for a vendor-defined Raw format.

storageList/type/format/reservedTemplate2

Reserved for a vendor-defined Template format.

storageList/type/format/reservedRaw3

Reserved for a vendor-defined Raw format.

storageList/type/format/reservedTemplate3

Reserved for a vendor-defined Template format.

storageList/type/algorithm

Specifies the encryption algorithm.

storageList/type/algorithm/ecb

Triple DES with Electronic Code Book.

storageList/type/algorithm/cbc

Triple DES with Cipher Block Chaining

storageList/type/algorithm/cfb

Triple DES with Cipher Feed Back.

storageList/type/algorithm/rsa

RSA Encryption.

storageList/type/keyName

Specifies the name of the key that is used to encrypt the biometric data. This value is omitted if the biometric data is not encrypted. The detailed key information is available through the [KeyManagement.GetKeyDetail](#).

Event Messages

None

[16.3.2 - Biometric.Read](#)

This command enables the device for biometric scanning, then captures and optionally returns biometric data. A [Biometric.PresentSubjectEvent](#) event will be sent to notify the client when it is ready to begin scanning and a [Biometric.SubjectDetectedEvent](#) event sent for each scanning attempt. The *numCaptures* input parameter specifies how many captures should be attempted, unless it is zero in which case the device itself will determine this. Once this command has successfully captured biometric raw data it will complete with Success.

The [Biometric.Read](#) command has two purposes:

Scanning: The biometric data that is captured into the device can be processed into biometric template data and returned as an output parameter for enrollment or storage elsewhere, e.g. on a server or smart card.

Matching: The biometric data that is captured into the device can be used for subsequent matching. Once data has been scanned into the device it can be compared to existing biometric templates that have been imported using the [Biometric.Import](#) command in order to allow verification or identification of an individual. The [Capabilities.matchsupported](#) capability indicates if the [Biometric.Match](#) command can be used for matching, otherwise the matching must be done externally, e.g. on a server or smart card.

In either case the data that has been scanned into the device will be persistent according to the current persistence mode as reported by the *dataPersistence* status field.

Command Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "type": [{ | array (object) | |
| "format": { | object | |
| "isoFid": false, | boolean | |
| "isoFmd": false, | boolean | |
| "ansiFid": false, | boolean | |
| "ansiFmd": false, | boolean | |
| "qso": false, | boolean | |
| "wso": false, | boolean | |
| "reservedRaw1": false, | boolean | |
| "reservedTemplate1": false, | boolean | |
| "reservedRaw2": false, | boolean | |
| "reservedTemplate2": false, | boolean | |
| "reservedRaw3": false, | boolean | |
| "reservedTemplate3": false | boolean | |
| }, | | |
| "algorithm": { | object | |
| "ecb": false, | boolean | |
| "cbc": false, | boolean | |
| "cfb": false, | boolean | |
| "rsa": false | boolean | |
| }, | | |
| "keyName": Add example to YAML | string | |

```
}],  
  "numCaptures": 0,          integer  
  "mode": {                 object  
    "scan": false,           boolean  
    "match": false          boolean  
  }  
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

type

Array of data, each data element of which represents the data type(s) in which the data should be returned in the ReadData output parameter. If no data is to be returned Types should be set to NULL. Single or multiple formats can be returned, or no data can be returned in the case where the scan is to be followed by a subsequent matching operation.

type/format

Specifies the format of the template data.

type/format/isoFid

Raw ISO FID format

type/format/isoFmd

ISO FMD template format

type/format/ansiFid

Raw ANSI FID format

type/format/ansiFmd

ANSI FMD template format

type/format/qso

Raw QSO image format

type/format/wso

WSQ image format

type/format/reservedRaw1

Reserved for a vendor-defined Raw format.

type/format/reservedTemplate1

Reserved for a vendor-defined Template format.

type/format/reservedRaw2

Reserved for a vendor-defined Raw format.

type/format/reservedTemplate2

Reserved for a vendor-defined Template format.

type/format/reservedRaw3

Reserved for a vendor-defined Raw format.

type/format/reservedTemplate3

Reserved for a vendor-defined Template format.

type/algorithm

Specifies the encryption algorithm.

type/algorithm/ecb

Triple DES with Electronic Code Book.

type/algorithm/cbc

Triple DES with Cipher Block Chaining

type/algorithm/cfb

Triple DES with Cipher Feed Back.

type/algorithm/rsa

RSA Encryption.

type/keyName

Specifies the name of the key that is used to encrypt the biometric data. This value is omitted if the biometric data is not encrypted. The detailed key information is available through the [KeyManagement.GetKeyDetail](#).

numCaptures

This field indicates the number of times to attempt capture of the biometric data from the subject. If this is zero, then the device determines how many attempts will be made. The maximum number of captures possible is indicated by the [Capabilities.maxCapture](#) capability.

mode

This optional field indicates the reason why the [Biometric.Read](#) command has been issued,

in order to allow for any necessary optimization.

mode/scan

The **Biometric.Read** command can be used to scan data only, for example to enroll a user or collect data for matching in an external biometric system.

mode/match

The **Biometric.Read** command can be used to scan data for a match operation using the **Biometric.Match** command.

Completion Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "readFailed", | string | |
| "dataList": [| array (object) | |
| "type": { | object | |
| "format": { | object | |
| "isoFid": false, | boolean | |
| "isoFmd": false, | boolean | |
| "ansiFid": false, | boolean | |
| "ansiFmd": false, | boolean | |
| "qso": false, | boolean | |
| "wso": false, | boolean | |
| "reservedRaw1": false, | boolean | |
| "reservedTemplate1": false, | boolean | |
| "reservedRaw2": false, | boolean | |

```

"reservedTemplate2": false,           boolean
"reservedRaw3": false,                boolean
"reservedTemplate3": false           boolean
},
"algorithm": {                       object
"ecb": false,                      boolean
"cbc": false,                      boolean
"cfb": false,                      boolean
"rsa": false,                      boolean
},
"keyName": Add example to YAML      string
},
"data": {                           object
"length": 0,                      integer
"data": {                          object
}
}
}
}
}

```

Properties

errorCode

Specifies the error code if applicable. The following values are possible:

- **readFailed** - Module was unable to complete the scan operation.
- **modeNotSupp** - The input parameter *Mode* contains a value that is not supported.
- **formatNotSupp** - The format specified is valid but not supported. A list of the supported values can be obtained through the [Capabilities.DataFormat](#) capability field.
- **keyNotFound** - The specified key name is not found.

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

dataList

Array of Data, used to contain the returned data and its format.

dataList/type

This field is used to indicate the biometric data type of the template data contained in Data.

dataList/type/format

Specifies the format of the template data.

dataList/type/format/isoFid

Raw ISO FID format

dataList/type/format/isoFmd

ISO FMD template format

dataList/type/format/ansiFid

Raw ANSI FID format

dataList/type/format/ansiFmd

ANSI FMD template format

dataList/type/format/qso

Raw QSO image format

dataList/type/format/wso

WSQ image format

dataList/type/format/reservedRaw1

Reserved for a vendor-defined Raw format.

dataList/type/format/reservedTemplate1

Reserved for a vendor-defined Template format.

dataList/type/format/reservedRaw2

Reserved for a vendor-defined Raw format.

dataList/type/format/reservedTemplate2

Reserved for a vendor-defined Template format.

dataList/type/format/reservedRaw3

Reserved for a vendor-defined Raw format.

dataList/type/format/reservedTemplate3

Reserved for a vendor-defined Template format.

dataList/type/algorithm

Specifies the encryption algorithm.

dataList/type/algorithm/ecb

Triple DES with Electronic Code Book.

dataList/type/algorithm/cbc

Triple DES with Cipher Block Chaining

dataList/type/algorithm/cfb

Triple DES with Cipher Feed Back.

dataList/type/algorithm/rsa

RSA Encryption.

dataList/type/keyName

Specifies the name of the key that is used to encrypt the biometric data. This value is omitted if the biometric data is not encrypted. The detailed key information is available through the [KeyManagement.GetKeyDetail](#).

dataList/data

It contains the binary data stream.

dataList/data/length

Length of the byte stream.

dataList/data/data

It contains the individual binary data stream

Event Messages

- [Biometric.PresentSubjectEvent](#)
- [Biometric.SubjectDetectedEvent](#)
- [Biometric.RemoveSubjectEvent](#)
- [Biometric.SubjectRemovedEvent](#)
- [Biometric.DataClearedEvent](#)
- [Biometric.OrientationEvent](#)

[16.3.3 - Biometric.Import](#)

This command imports a list of biometric template data structures into the device for later comparison with biometric data scanned using the [Biometric.Read](#) command. Normally this data is read from the chip on a customer's card or provided by the host system. Data that has been imported is available until a [Biometric.Clear](#) command is called. If template data has been previously imported using a call to [Biometric.Import](#), then it is overwritten. This data is not persistent across power fails.

Command Message

| Payload | Type | Required |
|--|----------------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "dataList": [{ | array (object) | |
| "type": { | object | |
| "format": { | object | |
| "isoFid": false, | boolean | |
| "isoFmd": false, | boolean | |
| "ansiFid": false, | boolean | |
| "ansiFmd": false, | boolean | |
| "qso": false, | boolean | |
| "wso": false, | boolean | |
| "reservedRaw1": false, | boolean | |
| "reservedTemplate1": false, | boolean | |
| "reservedRaw2": false, | boolean | |
| "reservedTemplate2": false, | boolean | |
| "reservedRaw3": false, | boolean | |
| "reservedTemplate3": false | boolean | |
| }, | | |
| "algorithm": { | object | |
| "ecb": false, | boolean | |
| "cbc": false, | boolean | |
| "cfb": false, | boolean | |
| "rsa": false | boolean | |
| }, | | |
| "keyName": Add example to YAML | string | |

```
},  
  "data": {  
    "length": 0,  
    "data": {  
      }  
    }  
  }]  
}
```

Properties

dataList

Array of Data. The data type is used to contain the returned data and its format.

dataList/type

This field is used to indicate the biometric data type of the template data contained in Data.

dataList/type/format

Specifies the format of the template data.

dataList/type/format/isoFid

Raw ISO FID format

dataList/type/format/isoFmd

ISO FMD template format

dataList/type/format/ansiFid

Raw ANSI FID format

`dataList/type/format/ansiFmd`

ANSI FMD template format

`dataList/type/format/qso`

Raw QSO image format

`dataList/type/format/wso`

WSQ image format

`dataList/type/format/reservedRaw1`

Reserved for a vendor-defined Raw format.

`dataList/type/format/reservedTemplate1`

Reserved for a vendor-defined Template format.

`dataList/type/format/reservedRaw2`

Reserved for a vendor-defined Raw format.

`dataList/type/format/reservedTemplate2`

Reserved for a vendor-defined Template format.

`dataList/type/format/reservedRaw3`

Reserved for a vendor-defined Raw format.

`dataList/type/format/reservedTemplate3`

Reserved for a vendor-defined Template format.

`dataList/type/algorithm`

Specifies the encryption algorithm.

dataList/type/algorithm/ecb

Triple DES with Electronic Code Book.

dataList/type/algorithm/cbc

Triple DES with Cipher Block Chaining

dataList/type/algorithm/cfb

Triple DES with Cipher Feed Back.

dataList/type/algorithm/rsa

RSA Encryption.

dataList/type/keyName

Specifies the name of the key that is used to encrypt the biometric data. This value is omitted if the biometric data is not encrypted. The detailed key information is available through the [KeyManagement.GetKeyDetail](#).

dataList/data

It contains the binary data stream.

dataList/data/length

Length of the byte stream.

dataList/data/data

It contains the individual binary data stream

Completion Message

| Payload | Type | Required |
|---------|------|----------|
| { | | |

```

"completionCode": "success",           string
"errorDescription": Add example to YAML,  string
"errorCode": "invalidData",           string
"storageList": {                      object
  "storageList": [{}                  array (object)
    "identifier": 0,                 integer
    "type": {                      object
      "format": {                  object
        "isoFid": false,            boolean
        "isoFmd": false,            boolean
        "ansiFid": false,           boolean
        "ansiFmd": false,           boolean
        "qso": false,                boolean
        "wso": false,                boolean
        "reservedRaw1": false,       boolean
        "reservedTemplate1": false,  boolean
        "reservedRaw2": false,       boolean
        "reservedTemplate2": false,  boolean
        "reservedRaw3": false,       boolean
        "reservedTemplate3": false   boolean
      },
      "algorithm": {                object
        "ecb": false,                boolean
        "cbc": false,                boolean
        "cfb": false,                boolean
        "rsa": false,                boolean
      },
    }
  ]
}

```

```
"keyName": Add example to YAML           string  
}  
}  
]  
"  
$ref":                                undefined  
}  
}  
}
```

Properties

errorCode

Specifies the error code if applicable. The following values are possible:

- invalidData - The data that was imported was malformed or invalid. No data has been imported into the device. The presence of any previously loaded templates can be checked for using the [Biometric.Read](#) command.
- formatNotSupp - The format of the biometric data that was specified is not supported. No data has been imported into the device. A list of the supported values can be obtained through the [Capabilities.DataFormat](#) capability field.
- capacityExceeded - An attempt has been made to import more templates than the maximum reserved storage space available. The maximum storage space available is reported in the capability [Capabilities.templateStorage](#). No data has been imported into the device. The amount of storage remaining is reported in the [Status.remainingStorage](#) status field.
- keyNotFound - The specified key name is not found.

storageList

A list of the biometric template data that were successfully imported.

storageList/storageList

Storage Array.

storageList/storageList/identifier

A unique number which identifies the template.

`storageList/storageList/type`

Specifies the biometric data type of the template data.

`storageList/storageList/type/format`

Specifies the format of the template data.

`storageList/storageList/type/format/isoFid`

Raw ISO FID format

`storageList/storageList/type/format/isoFmd`

ISO FMD template format

`storageList/storageList/type/format/ansiFid`

Raw ANSI FID format

`storageList/storageList/type/format/ansiFmd`

ANSI FMD template format

`storageList/storageList/type/format/qso`

Raw QSO image format

`storageList/storageList/type/format/wso`

WSQ image format

`storageList/storageList/type/format/reservedRaw1`

Reserved for a vendor-defined Raw format.

`storageList/storageList/type/format/reservedTemplate1`

Reserved for a vendor-defined Template format.

storageList/storageList/type/format/reservedRaw2

Reserved for a vendor-defined Raw format.

storageList/storageList/type/format/reservedTemplate2

Reserved for a vendor-defined Template format.

storageList/storageList/type/format/reservedRaw3

Reserved for a vendor-defined Raw format.

storageList/storageList/type/format/reservedTemplate3

Reserved for a vendor-defined Template format.

storageList/storageList/type/algorithm

Specifies the encryption algorithm.

storageList/storageList/type/algorithm/ecb

Triple DES with Electronic Code Book.

storageList/storageList/type/algorithm/cbc

Triple DES with Cipher Block Chaining

storageList/storageList/type/algorithm/cfb

Triple DES with Cipher Feed Back.

storageList/storageList/type/algorithm/rsa

RSA Encryption.

storageList/storageList/type/keyName

Specifies the name of the key that is used to encrypt the biometric data. This value is omitted if the biometric data is not encrypted. The detailed key information is available through the [KeyManagement.GetKeyDetail](#).

Event Messages

None

[16.3.4 - Biometric.Match](#)

This command returns the result of a comparison between data that has been scanned using the [Biometric.Read](#) command and template data that has been imported using the [Biometric.Import](#) command. The comparison may be performed by this command or the [Biometric.Read](#), this command is responsible for returning the result. Success is returned if the device has been able to successfully compare the data, however this does not necessarily mean that the data matched.

If the capability *matchSupported* value == CombinedMatch then the device performs a combined scan and match operation, and the [Biometric.SetMatch](#) must be called before this command in order to set the matching criteria. In this case if [Biometric.SetMatch](#) has not been called then this command will fail with SequenceError.

If the capability *matchSupported* == StoredMatch then the device will scan data using the [Biometric.Read](#) command and store it, then the data can be compared with imported biometric data using the [Biometric.Match](#) command.

This command can be used in two modes of operation: Verification or Identification, as indicated by the *compareMode* input parameter. The two modes of operation are described below:

Verification (*compareMode* == verify) :

In this case a one to one comparison is performed and the *maximum* input parameter is ignored. The data that has been scanned previously using the [Biometric.Read](#) command is compared with a single template that has been imported using the [Biometric.Import](#) command. If there is a successful match then the *ConfidenceLevel* output parameter can be used to determine the quality of the match and will be in the range 0 – 100, where 100 represents an exact match and 0 represents no match.

Identification (*compareMode* == identify) :

In this case a one to many comparison is performed. The data that has been scanned previously using the [Biometric.Read](#) command is compared with multiple templates that have been imported using the [Biometric.Import](#) command. The input parameter *maximum* is used to specify the maximum number of matches to return: a smaller number can make execution faster. The required degree of matching similarity can be controlled using the *threshold* parameter which is used to control the frequency of false positive and false

negative matching errors. The value of *threshold* represents the criteria as to what constitutes a successful match and is in the range 0 – 100, where 100 represents an exact match and 0 represents no match. If for example, *threshold* is set to 75 then only results with a matching score equal to or greater than 75 are returned. The matching candidate list is returned in the *matchResult* output parameter sorted in order of highest score. The higher the value of *confidenceLevel* the closer the candidate is to the beginning of the list, with the best match being the first candidate in the list. Note that where the number of templates that match the criteria of the threshold are greater than *maximum*, only the *maximum* templates with the highest score will be returned.

Command Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "compareMode": { | object | |
| "verify": false, | boolean | |
| "identity": false | boolean | |
| }, | | |
| "identifier": 0, | integer | |
| "maximum": 0, | integer | |
| "threshold": 0 | integer | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

compareMode

Specifies the type of match operation that is being done.

compareMode/verify

The biometric data can be compared as a one to one verification operation.

compareMode/identity

The biometric data can be compared as a one to many identification operation

identifier

In the case where *compareMode* is verify this parameter corresponds to a template that has been imported by a previous call to the [Biometric.Import](#) command. If *compareMode* is identify a comparison is performed against all of the imported templates, in which case this parameter is ignored.

maximum

Specifies the maximum number of matches to return. In the case where *compareMode* is verify this parameter is ignored.

threshold

Specifies the minimum matching confidence level necessary for the candidate to be included in the results. This value should be in the range of 0 to 100, where 100 represents an exact match and 0 represents no match.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "noImportedData", | string | |

```

"templateList": [{                                         array (object)
  "confidenceLevel": 0,                            integer
  "identifier": 0,                                integer
  "data": {                                         object
    "type": {                                         object
      "format": {                                     object
        "isoFid": false,                           boolean
        "isoFmd": false,                           boolean
        "ansiFid": false,                          boolean
        "ansiFmd": false,                          boolean
        "qso": false,                             boolean
        "wso": false,                             boolean
        "reservedRaw1": false,                      boolean
        "reservedTemplate1": false,                 boolean
        "reservedRaw2": false,                      boolean
        "reservedTemplate2": false,                 boolean
        "reservedRaw3": false,                      boolean
        "reservedTemplate3": false                  boolean
      },
      "algorithm": {                                object
        "ecb": false,                            boolean
        "cbc": false,                            boolean
        "cfb": false,                            boolean
        "rsa": false                            boolean
      },
      "keyName": Add example to YAML           string
    },
    "data": {                                         object

```

```

"length": 0,                                integer
"data": {                                    object
}
}
}
}
}]
}

```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- noImportedData - The command failed because no data was imported previously using the [ImportData](#) command.
- invalidIdentifier - The command failed because data was imported but *identifier* was not found.
- modeNotSupp - The type of match specified in *compareMode* is not supported.
- noCaptureData - No captured data is present. Typically means that the [Biometric.Read](#) command has not been called, or the captured data has been cleared using the [Biometric.Clear](#) command.
- invalidCompareMode - The compare mode specified by the *compareMode* input parameter is not supported.
- invalidThreshold - The *Threshold* input parameter is greater than the maximum allowed of 100.

templateList

Array of template. This will be an empty list if the [Biometric.Match](#) operation completes with no match found. If there are matches found, templateList contains all of the matching templates in order of confidence level, with the highest score first. Note that where the number of templates that match the input criteria of the threshold are greater than *maximum*, only the *maximum* templates with the highest scores will be returned.

templateList/confidenceLevel

Specifies the level of confidence for the match found. This value is in a scale of 0 - 100, where 0 is no match and 100 is an exact match. The minimum value will be that which was set by the Threshold input parameter.

templateList/identifier

A unique number that positively identifies the biometric template data. This corresponds to the list of template identifiers returned by the [Biometric.GetStorageInfo](#) command.

templateList/data

Contains the biometric template data that was matched. This data may be used as justification for the biometric data match or confidence level. Data is NULL if no additional comparison data is returned.

templateList/data/type

This field is used to indicate the biometric data type of the template data contained in Data.

templateList/data/type/format

Specifies the format of the template data.

templateList/data/type/format/isoFid

Raw ISO FID format

templateList/data/type/format/isoFmd

ISO FMD template format

templateList/data/type/format/ansiFid

Raw ANSI FID format

templateList/data/type/format/ansiFmd

ANSI FMD template format

templateList/data/type/format/qso

Raw QSO image format

templateList/data/type/format/wso

WSQ image format

templateList/data/type/format/reservedRaw1

Reserved for a vendor-defined Raw format.

templateList/data/type/format/reservedTemplate1

Reserved for a vendor-defined Template format.

templateList/data/type/format/reservedRaw2

Reserved for a vendor-defined Raw format.

templateList/data/type/format/reservedTemplate2

Reserved for a vendor-defined Template format.

templateList/data/type/format/reservedRaw3

Reserved for a vendor-defined Raw format.

templateList/data/type/format/reservedTemplate3

Reserved for a vendor-defined Template format.

templateList/data/type/algorithm

Specifies the encryption algorithm.

templateList/data/type/algorithm/ecb

Triple DES with Electronic Code Book.

templateList/data/type/algorithm/cbc

Triple DES with Cipher Block Chaining

templateList/data/type/algorithm/cfb

Triple DES with Cipher Feed Back.

templateList/data/type/algorithm/rsa

RSA Encryption.

templateList/data/type/KeyName

Specifies the name of the key that is used to encrypt the biometric data. This value is omitted if the biometric data is not encrypted. The detailed key information is available through the [KeyManagement.GetKeyDetail](#).

templateList/data/data

It contains the binary data stream.

templateList/data/data/length

Length of the byte stream.

templateList/data/data/data

It contains the individual binary data stream

Event Messages

- [Biometric.DataClearedEvent](#)

[16.3.5 - Biometric.SetMatch](#)

This command is used for devices which need to know the match criteria data for the [Biometric.Match](#) command before any biometric scanning is performed by the [Biometric.Read](#) command. [Biometric.Read](#) and [Biometric.Match](#) should be called after this command. For all other devices Unsupp_Command will be returned here.

If the capability *matchSupported* == CombinedMatch then this command is mandatory. If it is not called first, the [Biometric.Match](#) command will fail with the generic error SequenceError. The data set using this command is not persistent across power failures.

Command Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "compareMode": { | object | |
| "verify": false, | boolean | |
| "identity": false | boolean | |
| }, | | |
| "identifier": 0, | integer | |
| "maximum": 0, | integer | |
| "threshold": 0 | integer | |
| } | | |

*Properties***completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

compareMode

Specifies the type of match operation that is being done.

compareMode/verify

The biometric data can be compared as a one to one verification operation.

compareMode/identity

The biometric data can be compared as a one to many identification operation

identifier

In the case where *compareMode* is verify this parameter corresponds to a template that has been imported by a previous call to the [Biometric.Import](#) command. If *compareMode* is identify a comparison is performed against all of the imported templates, in which case this parameter is ignored.

maximum

Specifies the maximum number of matches to return. In the case where *compareMode* is verify this parameter is ignored.

threshold

Specifies the minimum matching confidence level necessary for the candidate to be included in the results. This value should be in the range of 0 to 100, where 100 represents an exact match and 0 represents no match.

Completion Message

Payload

Type Required

```
{  
  "completionCode": "success",          string  
  "errorDescription": Add example to YAML,  string  
  "errorCode": "invalidIdentifier"       string  
}
```

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

errorCode

Specifies the error code if applicable. The following values are possible:

- invalidIdentifier - The command failed because data was imported but *identifier* was not found.
- modeNotSupp - The type of match specified in *compareMode* is not supported.
- noImportedData - The command failed because no data was imported previously using the [Biometric.ImportData](#) command.
- invalidThreshold - The *threshold* input parameter is greater than the maximum allowed of 100.

[Event Messages](#)

None

[16.3.6 - Biometric.Clear](#)

This command can be used to clear stored data. In the case where there is no stored data to clear this command completes with Success.

Command Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "clearData": { | object | |
| "scannedData": false, | boolean | |
| "importedData": false, | boolean | |
| "setMatchedData": false | boolean | |
| } | | |
| } | | |

Properties**clearData**

This parameter indicates the type of data to be cleared from storage. If this is set to zero then all stored data will be cleared.

clearData/scannedData

Raw image data that has been scanned using the [Biometric.Read](#) command can be cleared

clearData/importedData

Template data that was imported using the [Biometric.Import](#) command can be cleared.

clearData/setMatchedData

Match criteria data that was set using the [Biometric.Match](#) command can be cleared.

Completion Message

| Payload | Type | Required |
|---|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML | string | |
| } | | |

Properties**completionCode**

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

Event Messages

- Biometric.DataClearedEvent

16.3.7 - Biometric.Reset

This command is used by the client to perform a hardware reset which will attempt to return the biometric device to a known good state.

Command Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "clearData": { | object | |
| "scannedData": false, | boolean | |

```

"importedData": false,           boolean
"setMatchedData": false          boolean
}
}

```

Properties

clearData

This parameter indicates the type of data to be cleared from storage. If this is set to zero then all stored data will be cleared.

clearData/scannedData

Raw image data that has been scanned using the [Biometric.Read](#) command can be cleared

clearData/importedData

Template data that was imported using the [Biometric.Import](#) command can be cleared.

clearData/setMatchedData

Match criteria data that was set using the [Biometric.Match](#) command can be cleared.

Completion Message

| Payload | Type | Required |
|---|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML | string | |
| } | | |

Properties

completionCode

success if the command was successful otherwise error

errorDescription

If not success, then this is optional vendor dependent information to provide additional information

Event Messages

- [Biometric.DataClearedEvent](#)

16.3.8 - Biometric.SetDataPersistance

This command is used to set the persistence mode. This controls how the biometric data is persisted after a [Biometric.Read](#) command. The data can be persisted for use by subsequent commands, or it can be automatically cleared.

Command Message

| Payload | Type | Required |
|--|---------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "persistanceMode": { | object | |
| "persist": false, | boolean | |
| "clear": false | boolean | |
| } | | |
| } | | |

Properties**persistanceMode**

Specifies the data persistence mode. This controls how biometric data that has been

All rights of exploitation in any form and by any means reserved worldwide for CEN
national Members.

captured using the [Biometric.Read](#) command will persist. This value itself is persistent.

`persistanceMode/persist`

Biometric data captured using the [Biometric.Read](#) command can persist until all sessions are closed, the device is power failed or rebooted, or the [Biometric.Read](#) command is requested again. This captured biometric data can also be explicitly cleared using the [Biometric.Clear](#) or [Biometric.Reset](#) commands.

`persistanceMode/clear`

Captured biometric data will not persist. Once the data has been either returned in the [Biometric.Read](#) command or used by the [Biometric.Match](#) command, then the data is cleared from the device.

Completion Message

| Payload | Type | Required |
|--|--------|----------|
| { | | |
| "completionCode": "success", | string | |
| "errorDescription": Add example to YAML, | string | |
| "errorCode": "modeNoSupp" | string | |
| } | | |

Properties

`completionCode`

success if the command was successful otherwise error

`errorDescription`

If not success, then this is optional vendor dependent information to provide additional information

`errorCode`

Specifies the error code if applicable. The following values are possible:

- modeNoSupp - The command failed because a mode was specified which is not supported.

Event Messages

None

16.4 - Unsolicited Messages

16.4.1 - Biometric.PresentSubjectEvent

This execute event is generated to notify the client when the device is ready for a user to present the subject to be captured to the biometric scanner, for example, placing a finger on a fingerprint reader.

16.4.2 - Biometric.SubjectDetectedEvent

This execute event is generated to notify the client when the device has detected a subject in the capture area and an attempt to capture biometric data has been performed.

16.4.3 - Biometric.RemoveSubjectEvent

This execute event is used to notify a client that the subject should be removed from the capture area of the device.

16.4.4 - Biometric.SubjectRemovedEvent

This service event is generated when the subject has been removed from the capture area of the device. This event may be generated at any time.

16.4.5 - Biometric.DataClearedEvent

This mandatory event notifies the client when data has been cleared. This can be the case when the data is cleared automatically after a [Biometric.Read](#) or [Biometric.Match](#) command completion, or as a result of an explicit call to the [Biometric.Clear](#) or [Biometric.Reset](#) commands.

| Payload | Type | Required |
|-------------------------|---------|----------|
| { | | |
| "clearData": { | object | |
| "scannedData": false, | boolean | |
| "importedData": false, | boolean | |
| "setMatchedData": false | boolean | |
| } | | |
| } | | |

Properties

clearData

This parameter indicates the data that was cleared from storage.

clearData/scannedData

Raw image data that has been scanned using the [Biometric.Read](#) command can be cleared

clearData/importedData

Template data that was imported using the [Biometric.Import](#) command can be cleared.

clearData/setMatchedData

Match criteria data that was set using the [Biometric.Match](#) command can be cleared.

16.4.6 - Biometric.OrientationEvent

This event is generated when the biometric subject has an incorrect orientation relative to the device scanner in order to allow a client to prompt a user to correct it.

XFS4IoT Preview version 0.3. Next preview - Q1 2021. Note: work-in-progress. Use at your own risk.

All rights of exploitation in any form and by any means reserved worldwide for CEN national members.