Extensions for Financial Services (XFS)
XFS4IoT Specification Preview 0.4

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.

Warning.

This document is not a CEN Workshop Agreement. It is distributed for review and comment. It is subject to change without notice and may not be referred to as a CEN Workshop Agreement.

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.

Table of Contents

1. API 16
1.1 Summary 16
1.2 General Information 16
1.2.1 Messages 16
1.2.2 Command Sequence 18
1.2.3 Message Versions 23
1.2.4 End to End Security 28
2. Service Publisher 29
2.1 Summary 29
2.2 General Information 29
2.2.1 Overview 29
2.2.2 Machine Identification 29
2.2.3 Network Protocol 29
2.2.4 URI Format 30
2.2.5 Service Publishing 31
2.2.6 Port Sequence 31
2.2.7 Free End-point Port Discovery 31
2.2.8 Handling Incoming Connections 31
2.2.9 Client 32
2.2.10 Publisher End-point Discovery 32
2.2.11 Service Endpoint Discovery 32
2.3 Command Messages 35
2.3.1 ServicePublisher.GetServices 35
2.4 Event Messages 37
2.4.1 ServicePublisher.ServiceDetailEvent 37
3. Common Interface 38
3.1 Summary 38
3.2 Command Messages 39
3.2.1 Common.Status 39
3.2.2 Common.Capabilities 71
3.2.3 Common.SetVersions 152
3.2.4 Common.Cancel 154
3.2.5 Common.PowerSaveControl 156
3.2.6 Common.SynchronizeCommand 157
3.2.7 Common.SetTransactionState 159

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

3.2.8 Common.GetTransactionState 160
3.2.9 Common.GetCommandNonce 161
3.2.10 Common.ClearCommandNonce 162
3.3 Unsolicited Messages 163
3.3.1 Common.PowerSaveChangeEvent 163
3.3.2 Common.DevicePositionEvent 164
3.3.3 Common.NonceClearedEvent 165
3.3.4 Common.ExchangeStateChangedEvent 166
4. Card Reader Interface 167
4.1 Summary 167
4.2 General Information 168
4.2.1 References 168
4.2.2 Intelligent Contactless Card Reader 168
4.3 Command Messages 173
4.3.1 CardReader.QueryIFMIdentifier 173
4.3.2 CardReader.EMVClessQueryApplications 175
4.3.3 CardReader.ReadRawData 177
4.3.4 CardReader.WriteRawData 184
4.3.5 CardReader.Move 187
4.3.6 CardReader.SetKey 189
4.3.7 CardReader.ChipIO 190
4.3.8 CardReader.Reset 192
4.3.9 CardReader.ChipPower 194
4.3.10 CardReader.EMVClessConfigure 196
4.3.11 CardReader.EMVClessPerformTransaction 199
4.3.12 CardReader.EMVClessIssuerUpdate 205
4.4 Event Messages 210
4.4.1 CardReader.InsertCardEvent 210
4.4.2 CardReader.MediaInsertedEvent 211
4.4.3 CardReader.InvalidMediaEvent 212
4.4.4 CardReader.TrackDetectedEvent 213
4.4.5 CardReader.InvalidTrackDataEvent 214
4.4.6 Storage.StorageThresholdEvent 215
4.4.7 CardReader.MediaDetectedEvent 227
4.4.8 CardReader.EMVClessReadStatusEvent 228
4.5 Unsolicited Messages 230
4.5.1 CardReader.MediaRemovedEvent 230

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

4.5.2 CardReader.CardActionEvent 231
5. Cash Management Interface 232
5.1 Summary 232
5.2 General Information 232
5.2.1 Note Classification 232
5.3 Command Messages 234
5.3.1 CashManagement.GetCashUnitInfo 234
5.3.2 CashManagement.GetTellerInfo 240
5.3.3 CashManagement.GetItemInfo 243
5.3.4 CashManagement.GetClassificationList 247
5.3.5 CashManagement.SetTellerInfo 249
5.3.6 CashManagement.SetCashUnitInfo 252
5.3.7 CashManagement.OpenSafeDoor 258
5.3.8 CashManagement.StartExchange 259
5.3.9 CashManagement.EndExchange 267
5.3.10 CashManagement.CalibrateCashUnit 269
5.3.11 CashManagement.SetClassificationList 273
5.4 Event Messages 275
5.4.1 CashManagement.CashUnitErrorEvent 275
5.4.2 CashManagement.NoteErrorEvent 281
5.4.3 CashManagement.InfoAvailableEvent 282
5.4.4 CashAcceptor.ShutterStatusChangedEvent 283
5.4.5 CashDispenser.ItemsTakenEvent 284
5.5 Unsolicited Messages 285
5.5.1 CashManagement.SafeDoorOpenEvent 285
5.5.2 CashManagement.SafeDoorClosedEvent 286
5.5.3 CashManagement.CashUnitInfoChangedEvent 287
5.5.4 CashManagement.TellerInfoChangedEvent 293
5.5.5 CashManagement.CashUnitThresholdEvent 294
6. Cash Dispenser Interface 299
6.1 Summary 299
6.2 Command Messages 300
6.2.1 CashDispenser.GetMixTypes 300
6.2.2 CashDispenser.GetMixTable 302
6.2.3 CashDispenser.GetPresentStatus 304
6.2.4 CashDispenser.Denominate 308
6.2.5 CashDispenser.Dispense 311

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

6.2.6 CashDispenser.Present 317
6.2.7 CashDispenser.Reject 320
6.2.8 CashDispenser.Retract 321
6.2.9 CashDispenser.OpenShutter 324
6.2.10 CashDispenser.CloseShutter 326
6.2.11 CashDispenser.SetMixTable 328
6.2.12 CashDispenser.Reset 330
6.2.13 CashDispenser.TestCashUnits 333
6.2.14 CashDispenser.Count 341
6.2.15 CashDispenser.PrepareDispense 344
6.3 Event Messages 346
6.3.1 CashManagement.CashUnitErrorEvent 346
6.3.2 CashManagement.CashUnitThresholdEvent 352
6.3.3 CashDispenser.DelayedDispenseEvent 357
6.3.4 CashDispenser.StartDispenseEvent 358
6.3.5 CashDispenser.PartialDispenseEvent 359
6.3.6 CashDispenser.SubDispenseOkEvent 360
6.3.7 CashDispenser.IncompleteDispenseEvent 361
6.3.8 CashManagement.NoteErrorEvent 362
6.3.9 CashManagement.InfoAvailableEvent 363
6.3.10 CashDispenser.IncompleteRetractEvent 364
6.3.11 CashManagement.CashUnitInfoChangedEvent 366
6.4 Unsolicited Messages 372
6.4.1 CashDispenser.ItemsTakenEvent 372
6.4.2 CashDispenser.ItemsPresentedEvent 373
6.4.3 CashDispenser.MediaDetectedEvent 374
6.4.4 CashDispenser.ShutterStatusChangedEvent 376
7. Cash Acceptor Interface 377
7.1 Summary 377
7.2 Command Messages 378
7.2.1 CashAcceptor.GetBanknoteTypes 378
7.2.2 CashAcceptor.GetCashInStatus 380
7.2.3 CashAcceptor.GetPositionCapabilities 382
7.2.4 CashAcceptor.GetReplenishTarget 385
7.2.5 CashAcceptor.GetDevicelockStatus 386
7.2.6 CashAcceptor.GetCashUnitCapabilities 388
7.2.7 CashAcceptor.GetDepleteSource 391

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

7.2.8 CashAcceptor.GetCashUnitCountStatus 392
7.2.9 CashAcceptor.GetPresentStatus 394
7.2.10 CashAcceptor.CashInStart 397
7.2.11 CashAcceptor.CashIn 399
7.2.12 CashAcceptor.CashInEnd 402
7.2.13 CashAcceptor.CashInRollback 409
7.2.14 CashAcceptor.Retract 416
7.2.15 CashAcceptor.OpenShutter 423
7.2.16 CashAcceptor.CloseShutter 425
7.2.17 CashAcceptor.Reset 427
7.2.18 CashAcceptor.ConfigureNotetypes 430
7.2.19 CashAcceptor.CreateP6Signature 431
7.2.20 CashAcceptor.ConfigureNoteReader 434
7.2.21 CashAcceptor.CompareP6Signature 435
7.2.22 CashAcceptor.Replenish 438
7.2.23 CashAcceptor.SetCashInLimit 441
7.2.24 CashAcceptor.CashUnitCount 443
7.2.25 CashAcceptor.DeviceLockControl 445
7.2.26 CashAcceptor.SetMode 447
7.2.27 CashAcceptor.PresentMedia 448
7.2.28 CashAcceptor.Deplete 450
7.2.29 CashAcceptor.PreparePresent 453
7.3 Event Messages 455
7.3.1 CashManagement.CashUnitErrorEvent 455
7.3.2 CashAcceptor.InputRefuseEvent 461
7.3.3 CashManagement.NoteErrorEvent 462
7.3.4 CashAcceptor.SubCashInEvent 463
7.3.5 CashManagement.InfoAvailableEvent 464
7.3.6 CashAcceptor.InsertItemsEvent 465
7.3.7 CashManagement.CashUnitThresholdEvent 466
7.3.8 CashManagement.CashUnitInfoChangedEvent 471
7.3.9 CashAcceptor.IncompleteReplenishEvent 477
7.3.10 CashAcceptor.IncompleteDepleteEvent 478
7.4 Unsolicited Messages 479
7.4.1 CashAcceptor.ItemsTakenEvent 479
7.4.2 CashAcceptor.ItemsPresentedEvent 480
7.4.3 CashAcceptor.ItemsInsertedEvent 481

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

7.4.4 CashAcceptor.MediaDetectedEvent 482
7.4.5 CashAcceptor.ShutterStatusChangedEvent 484
7.4.6 CashAcceptor.CountAccuracyChangedEvent 485
8. Key Management Interface 486
8.1 Summary 486
8.2 General Information 486
8.2.1 RKL Terminology 486
8.2.2 Remote Key Loading Using Signatures 488
8.2.3 Remote Key Loading Using Certificates 496
8.2.4 Remote Key Loading Using TR34 499
8.2.5 EMV Support 502
8.2.6 ImportKey command Input-Output Parameters 504
8.2.7 TR-31 Key Use 509
8.2.8 RestrictedKeyEndKey Command Usage 510
8.3 Command Messages 516
8.3.1 KeyManagement.GetKeyDetail 516
8.3.2 KeyManagement.Initialization 523
8.3.3 KeyManagement.DeriveKey 526
8.3.4 KeyManagement.Reset 528
8.3.5 KeyManagement.ImportKey 529
8.3.6 KeyManagement.DeleteKey 538
8.3.7 KeyManagement.ExportRSAIssuerSignedItem 540
8.3.8 KeyManagement.GenerateRSAKeyPair 542
8.3.9 KeyManagement.ExportRSAEPPSignedItem 544
8.3.10 KeyManagement.GetCertificate 546
8.3.11 KeyManagement.ReplaceCertificate 548
8.3.12 KeyManagement.StartKeyExchange 550
8.3.13 KeyManagement.GenerateKCV 552
8.3.14 KeyManagement.LoadCertificate 554
8.3.15 KeyManagement.StartAuthenticate 556
8.4 Unsolicited Messages 558
8.4.1 KeyManagement.InitializedEvent 558
8.4.2 KeyManagement.IllegalKeyAccessEvent 559
8.4.3 KeyManagement.CertificateChangeEvent 560
9. Crypto Interface 561
9.1 Summary 561
9.2 General Information 561

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

9.2.1 DUKPT 561
9.3 Command Messages 562
9.3.1 Crypto.GenerateRandom 562
9.3.2 Crypto.CryptoData 564
9.3.3 Crypto.GenerateAuthentication 567
9.3.4 Crypto.VerifyAuthentication 570
9.3.5 Crypto.Digest 573
9.4 Event Messages 575
9.4.1 Crypto.DUKPTKSNEvent 575
9.5 Unsolicited Messages 576
9.5.1 Crypto.IllegalKeyAccessEvent 576
10. Keyboard Interface 577
10.1 Summary 577
10.2 General Information 577
10.2.1 Encrypting Touch Screen (ETS) 577
10.2.2 Layout 580
10.2.3 Command Usage 581
10.3 Command Messages 583
10.3.1 Keyboard.GetLayout 583
10.3.2 Keyboard.PinEntry 591
10.3.3 Keyboard.DataEntry 595
10.3.4 Keyboard.Reset 599
10.3.5 Keyboard.SecureKeyEntry 600
10.3.6 Keyboard.KeypressBeep 605
10.3.7 Keyboard.DefineLayout 607
10.4 Event Messages 615
10.4.1 Keyboard.KeyEvent 615
10.4.2 Keyboard.EnterDataEvent 616
10.4.3 Keyboard.LayoutEvent 617
11. PinPad Interface 624
11.1 Summary 624
11.2 General Information 624
11.2.1 DUKPT 624
11.2.2 Default Keyname 624
11.3 Command Messages 625
11.3.1 PinPad.GetQueryPCIPTSDeviceId 625
11.3.2 PinPad.LocalDES 627

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

11.3.3 PinPad.LocalVisa 630
11.3.4 PinPad.PresentIDC 632
11.3.5 PinPad.Reset 634
11.3.6 PinPad.MaintainPin 635
11.3.7 PinPad.SetPinBlockData 636
11.3.8 PinPad.GetPinBlock 640
11.4 Event Messages 644
11.4.1 PinPad.DUKPTKSNEvent 644
11.5 Unsolicited Messages 645
11.5.1 PinPad.IllegalKeyAccessEvent 645
12. Printer Interface 646
12.1 Summary 646
12.2 General Information 646
12.2.1 Banking Printer Types 646
12.2.2 Forms Model 647
12.2.3 Command Overview 647
12.2.4 Form, Sub-Form, Field, Frame, Table and Media Definitions 648
12.2.5 Command and Event Flows during Single and Multi-Page / Wad Printing 668
12.3 Command Messages 673
12.3.1 Printer.GetFormList 673
12.3.2 Printer.GetMediaList 674
12.3.3 Printer.GetQueryForm 675
12.3.4 Printer.GetQueryMedia 678
12.3.5 Printer.GetQueryField 682
12.3.6 Printer.GetCodelineMapping 685
12.3.7 Printer.ControlMedia 687
12.3.8 Printer.PrintForm 691
12.3.9 Printer.ReadForm 697
12.3.10 Printer.RawData 701
12.3.11 Printer.MediaExtents 703
12.3.12 Printer.ResetCount 705
12.3.13 Printer.ReadImage 706
12.3.14 Printer.Reset 710
12.3.15 Printer.RetractMedia 712
12.3.16 Printer.DispensePaper 714
12.3.17 Printer.PrintRawFile 716
12.3.18 Printer.LoadDefinition 720

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

12.3.19 Printer.SupplyReplenish 722
12.3.20 Printer.ControlPassbook 724
12.3.21 Printer.SetBlackMarkMode 726
12.4 Event Messages 727
12.4.1 Printer.MediaPresentedEvent 727
12.4.2 Printer.NoMediaEvent 728
12.4.3 Printer.MediaInsertedEvent 729
12.4.4 Printer.FieldErrorEvent 730
12.4.5 Printer.FieldWarningEvent 731
12.4.6 Printer.MediaRejectedEvent 732
12.5 Unsolicited Messages 733
12.5.1 Printer.MediaTakenEvent 733
12.5.2 Printer.MediaInsertedUnsolicitedEvent 734
12.5.3 Printer.MediaPresentedUnsolicitedEvent 735
12.5.4 Printer.MediaDetectedEvent 736
12.5.5 Printer.RetractBinStatusEvent 737
12.5.6 Printer.DefinitionLoadedEvent 738
12.5.7 Printer.MediaAutoRetractedEvent 739
12.5.8 Printer.RetractBinThresholdEvent 740
12.5.9 Printer.PaperThresholdEvent 741
12.5.10 Printer.TonerThresholdEvent 742
12.5.11 Printer.LampThresholdEvent 743
12.5.12 Printer.InkThresholdEvent 744
13. Text Terminal Interface 745
13.1 Summary 745
13.2 General Information 745
13.2.1 References 745
13.2.2 Form and Field Definitions 745
13.3 Command Messages 749
13.3.1 TextTerminal.GetFormList 749
13.3.2 TextTerminal.GetQueryForm 750
13.3.3 TextTerminal.GetQueryField 752
13.3.4 TextTerminal.GetKeyDetail 755
13.3.5 TextTerminal.Beep 757
13.3.6 TextTerminal.ClearScreen 759
13.3.7 TextTerminal.SetResolution 761
13.3.8 TextTerminal.WriteForm 763

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

13.3.9 TextTerminal.ReadForm 765
13.3.10 TextTerminal.Write 767
13.3.11 TextTerminal.Read 769
13.3.12 TextTerminal.Reset 773
13.3.13 TextTerminal.DefineKeys 774
13.4 Unsolicited Messages 777
13.4.1 TextTerminal.FieldErrorEvent 777
13.4.2 TextTerminal.FieldWarningEvent 778
13.4.3 TextTerminal.KeyEvent 779
14. Barcode Reader Interface 780
14.1 Summary 780
14.2 Command Messages 781
14.2.1 BarcodeReader.Read 781
14.2.2 BarcodeReader.Reset 788
15. Card Embosser Interface 789
15.1 Summary 789
15.2 General Information 789
15.2.1 Embossing Form, Field and Media Definitions 789
15.3 Command Messages 794
15.3.1 CardEmbosser.GetFormList 794
15.3.2 CardEmbosser.GetQueryForm 795
15.3.3 CardEmbosser.GetMediaList 797
15.3.4 CardEmbosser.GetQueryMedia 798
15.3.5 CardEmbosser.GetQueryField 801
15.3.6 CardEmbosser.EmbossCard 803
15.3.7 CardEmbosser.Reset 806
15.3.8 CardEmbosser.SupplyReplenish 808
15.4 Event Messages 810
15.4.1 CardEmbosser.EmbossFailureEvent 810
15.4.2 CardEmbosser.FieldErrorEvent 811
15.4.3 CardEmbosser.FieldWarningEvent 812
15.5 Unsolicited Messages 813
15.5.1 CardEmbosser.InputBinThresholdEvent 813
15.5.2 CardEmbosser.OutputBinThresholdEvent 814
15.5.3 CardEmbosser.RetainBinThresholdEvent 815
15.5.4 CardEmbosser.MediaRemovedEvent 816
15.5.5 CardEmbosser.MediaDetectedEvent 817

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

15.5.6 CardEmbosser.TonerThresholdEvent 818
16. Biometric Interface 819
16.1 Summary 819
16.2 General Information 819
16.2.1 References 819
16.2.2 Enrollment 819
16.2.3 Biometric Matching 819
16.2.4 Biometric Device Types 820
16.2.5 Biometric Data Security 821
16.2.6 Biometric Device Command Flows 821
16.3 Command Messages 826
16.3.1 Biometric.GetStorageInfo 826
16.3.2 Biometric.Read 828
16.3.3 Biometric.Import 832
16.3.4 Biometric.Match 836
16.3.5 Biometric.SetMatch 840
16.3.6 Biometric.Clear 842
16.3.7 Biometric.Reset 843
16.3.8 Biometric.SetDataPersistance 844
16.4 Unsolicited Messages 846
16.4.1 Biometric.PresentSubjectEvent 846
16.4.2 Biometric.SubjectDetectedEvent 847
16.4.3 Biometric.RemoveSubjectEvent 848
16.4.4 Biometric.SubjectRemovedEvent 849
16.4.5 Biometric.DataClearedEvent 850
16.4.6 Biometric.OrientationEvent 851
17. Camera Interface 852
17.1 Summary 852
17.2 Command Messages 853
17.2.1 Camera.TakePicture 853
17.2.2 Camera.Reset 855
17.3 Event Messages 856
17.3.1 Camera.InvalidDataEvent 856
17.4 Unsolicited Messages 857
17.4.1 Camera.MediaThresholdEvent 857
18. German DK(ZKA) Interface 858

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

18.1 Summary ... 858
18.2 General Information ... 858
18.2.1 References ... 858
18.2.2 German ZKA GeldKarte (Deutsche Kreditwirtschaft) ... 858
18.3 Command Messages ... 878
18.3.1 DK.GetHSMTData ... 878
18.3.2 DK.GetQueryLogicalHSMDetail ... 879
18.3.3 DK.HSMSetTData ... 881
18.3.4 DK.SecureMsgSend ... 884
18.3.5 DK.SecureMsgReceive ... 886
18.3.6 DK.GetJournal ... 888
18.3.7 DK.HSMInit ... 890
18.3.8 DK.SetLogicalHSM ... 892
18.4 Unsolicited Messages ... 894
18.4.1 DK.HSMTDataChangedEvent ... 894
18.4.2 DK.HSMChangedEvent ... 895
18.4.3 DK.OPTRequiredEvent ... 896
19. Lights Interface ... 897
19.1 Summary ... 897
19.2 Command Messages ... 898
19.2.1 Lights.SetLight ... 898
20. Auxiliaries Interface ... 903
20.1 Summary ... 903
20.2 Command Messages ... 904
20.2.1 Auxiliaries.GetAutoStartupTime ... 904
20.2.2 Auxiliaries.ClearAutoStartupTime ... 907
20.2.3 Auxiliaries.Register ... 908
20.2.4 Auxiliaries.SetAuxiliaries ... 913
20.2.5 Auxiliaries.SetAutostartupTime ... 918
20.3 Unsolicited Messages ... 921
20.3.1 Auxiliaries.AuxiliaryStatusEvent ... 921
21. Storage Interface ... 930
21.1 Summary ... 930
21.2 General Information ... 930
21.2.1 Transaction Flows ... 930
21.3 Command Messages ... 933

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

21.3.1 Storage.GetStorage 933
21.3.2 Storage.SetStorage 945
21.3.3 Storage.StartExchange 952
21.3.4 Storage.EndExchange 954
21.4 Event Messages 956
21.4.1 Common.ExchangeStateChangedEvent 956
21.5 Unsolicited Messages 957
21.5.1 Storage.StorageChangedEvent 957
21.5.2 Storage.StorageThresholdEvent 969
21.5.3 Storage.StorageErrorEvent 981
22. Vendor Mode Interface 993
22.1 Summary 993
22.2 General Information 993
22.2.1 Vendor Mode 993
23. Vendor Mode 994
23.1 Vendor Mode Entry triggered by an XFS Application 994
23.2 Vendor Mode Exit 995
23.3 Command Messages 996
23.3.1 VendorMode.Register 996
23.3.2 VendorMode.EnterModeRequest 997
23.3.3 VendorMode.EnterModeAcknowledge 998
23.3.4 VendorMode.ExitModeRequest 999
23.3.5 VendorMode.ExitModeAcknowledge 1000
23.4 Unsolicited Messages 1001
23.4.1 VendorMode.EnterModeRequestEvent 1001
23.4.2 VendorMode.ExitModeRequestEvent 1002
23.4.3 VendorMode.ModeEnteredEvent 1003
23.4.4 VendorMode.ModeExitedEvent 1004
24. Vendor Application Interface 1005
24.1 Summary 1005
24.2 General Information 1005
24.2.1 Vendor Application 1005
25. Vendor Application 1006
25.1 Command Messages 1007
25.1.1 VendorApplication.StartLocalApplication 1007
25.1.2 VendorApplication.GetActiveInterface 1009

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

25.1.3 VendorApplication.SetActiveInterface 1010
25.2 Event Messages 1011
25.2.1 VendorApplication.InterfaceChangedEvent 1011
25.3 Unsolicited Messages 1012
25.3.1 VendorApplication.VendorAppExited 1012
25.3.2 VendorApplication.InterfaceChanged 1013
26. 3.x Migration 1014
26.1 Summary 1014
26.2 General Information 1014
26.2.1 CDM (Cash Dispense Module) 1014
26.2.2 CIM (Cash-In Module) 1016

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

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 Messages

Overview of the general message handling process for XFS4IoT.

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 Client	Intermediate message from the XFS4IoT Service indicating 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": {
    }
}
```

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

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": {
    }
}
```

Payload Definition

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

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. Dependance 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.

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

Example Command Request Message Sequence

1.2.2 Command Sequence

Overview of the general sequencing of XFS4IoT command messages.

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

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 unsolic 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.

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

Message Types

The normal command message sequence will be:

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

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.

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

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.

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.*

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.

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.

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

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.

Completion Codes

After a command message has been received and associated acknowledge sent, the completion code, either success or an error code, will be included in the completion message for that command. The interface specification will define command specific error codes that are valid for each completion message. No other error codes will be returned by the service for the completion message.

The completion message payload *completionCode* property contains one of the following values:

- `success` - success.
- `commandErrorCode` - Check the errorCode property for the command specific error code.
- `canceled` - Canceled using the Common.Cancel command.
- `timeOut` - Timed out after the client specified timeout.
- `deviceNotReady` - The device is not ready or timed out.
- `hardwareError` - An error occurred on the device.
- `internalError` - An internal inconsistency or other unexpected error occurred.
- `invalidCommand` - The command is not supported by the service.
- `invalidRequestID` - The request is invalid.
- `unsupportedCommand` - The command is valid for the interface, but is not supported by the service or device.
- `invalidData` - The command message contains invalid data.
- `userError` - The user is preventing proper operation of the device.
- `unsupportedData` - The command message contains data that is valid for the interface command, but is not supported by the service or device.
- `fraudAttempt` - The user is attempting a fraudulent act on the device.
- `sequenceError` - The command request is not valid at this time or in the devices current state.
- `authorisationRequired` - The command request cannot be performed because it requires authentication.
- `noCommandNonce` - The value of the nonce stored in the hardware was cleared by a powerfailure. The security token is invalid and the operation has been rejected.
- `invalidToken` - The security token is invalid and the operation has been rejected. There may be more details in the errorDescription text.
- `invalidTokenNonce` - The value of the nonce in the security token does not match the stored value. The security token is invalid and the operation has been rejected.
- `invalidTokenHMAC` - The value of the HMAC in the security token is incorrect. The security token is invalid and the operation has been rejected.
- `invalidTokenFormat` - The token format version value is not recognized, or the token format is somehow invalid. The security token is invalid and the operation has been rejected.
- `invalidTokenKeyNoValue` - The key used for the HMAC for a token has not been loaded and the token cannot be validated. The security token is invalid and the operation has been rejected.

When an error occurs, optional vendor specific information may be included in the *errorDescription* property.

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.

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

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.

Command Queueing

Some commands can be executed in parallel. For example, a status command that returns the current status can always be executed immediately event 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 synchronize status information with ongoing actions.

When it's not possible to execute a command immediately then commands will be queued and executed as soon as possible.

The acknowledge message is always sent before the command is queued.

Queued commands will normally be dequeued and executed in the order received. It is valid to execute queued commands in a different order to that received.

If the condition that caused a command to be queued clears, the command nearest the front of the queue that is blocked by that condition will be dequeued and executed ahead of any other commands nearer the front of the queue.

For example, if while idle, an Encrypting Pin Pad service receives the following command requests in the order listed:

1. Keyboard.DataEntry
2. Crypto.CryptoData
3. Keyboard.PinEntry
4. Crypto.Digest

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

The Service executes in parallel the *Keyboard.DataEntry* and *Crypto.CryptoData* commands as one uses the Pin Pad and the other uses the encryptor. The *Keyboard.PinEntry* and *Crypto.Digest* commands are added to the queue in that order. If the *Crypto.CryptData* command completes before the *Keyboard.DataEntry* command, the service will execute the *Crypto.Digest* command as the encryptor is available while keeping the *Keyboard.PinEntry* command on the queue as the Pin Pad is still in use by the *Keyboard.DataEntry* command.

The order of execution would therefore be:

1. Keyboard.DataEntry
2. Crypto.CryptoData
3. Crypto.Digest
4. Keyboard.PinEntry

Cancelation

A client can use the Common.Cancel command to attempt cancelation of one, multiple or all queued or executing commands at any time.

The Common.Cancel command adheres to the standard command message flow. That is, the Client must assign it a unique request ID when sending the command message, and the service will send both acknowledge and completion messages using that request ID. The Service will not send any event messages related to the Common.Cancel command request ID.

The Common.Cancel command can only be used to cancel request IDs associated with the client connection on which the command is sent. That is, one client cannot cancel another clients requests.

The Common.Cancel command itself can not be canceled. The Service will ignore any requestIDs that are Common.Cancel command requests.

Similarly, the Service will ignore requests IDs that do not match a queued or executing command request ID.

The Common.Cancel will complete immediately. It will not wait until the completion messages of the specified request IDs have been sent.

Completion of the Common.Cancel command does not imply when the commands requested to cancel, actually complete. Nor does it imply those commands will complete with completionCode *canceled*.

Clients should expect that, at some future point, commands may complete with a completionCode other than *canceled*. For example, device state prevents the command canceling forcing it to complete as if no cancel request had been received.

The Service will always cancel queued commands which have not started executing.

The Service must send completion messages, for any commands requests being canceled, after the completion message for the Common.Cancel command has been sent.

The Client should not attempt to cancel any one request ID more than once as it is the responsibility of the Service to maintain the cancel requested state of a command until the command completes. Sending multiple requests to cancel the same command will have no effect.

1.2.3 Message Versions

The messages types, command, event and unsolicited are assigned version numbers to enable evolution of individual messages. The completion message type inherits its version number from the command message it is associated with.

If a new version of a command message has a property which has an associated capability property, the service must implement, at a minimum, the version of the Common.Capabilities command that includes the associated capability property. This will allow the client to decide whether or not to use the command message property and the value it should be set to.

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

Each release of the specification defines the message version numbers of the command, event, completion and unsolicited messages included in that release of the specification. The specification number is different from the message version numbers. If a message definition does not change from one release of the specification to the next, the message version number will remain the same.

Version Numbers

Message version numbers have the form X.Y where X and Y are non-negative integers, and do not contain leading zeroes. X is the major version and Y is the minor version. The major version must be greater than 0 and will always increment. The minor version will increment while the major version remains the same. When the major version increments, the minor version resets to 0. For example, 1.1 -> 1.2 -> ... -> 1.10 -> 2.0.

Major Version Numbers

Major version X (X.y) numbers will be incremented in the specification if any backwards incompatible changes are introduced to the command, event, unsolicited or completion messages. It may also include minor level changes.

Major version increments represent a new command, event or unsolicited message. While there will likely be similarities with the previous major version, this is not guaranteed. It is anticipated that given the flexibility of JSON, major version increments will rarely be required.

Major version increments allow:

- Removal of command message properties.
- Change of definition of command message properties.
- Change of definition of completion message properties.
- Change of definition of event message properties.
- New event messages which cannot be ignored by the client.

Minor Version Numbers

Minor version Y (x.Y) numbers will be incremented in the specification if new, backwards compatible functionality is introduced to the command, event, completion or unsolicited message. It will also be incremented if any message property is marked as deprecated. It may be incremented if substantial new functionality or improvements are introduced where backwards compatibility is maintained.

Minor version increments allow:

- Additional command message properties.
- Additional completion, event and unsolicited message properties.
- New event messages which can be ignored by the client.

Additional command message properties must be optional. If omitted, the command behavior must be as defined in minor version 0 of the major version of the command message. If included, additional properties may change the behavior of the command. Clients that included additional command message properties that change behavior should therefore handle these behavioral changes.

For additional completion, event and unsolicited message properties, clients should expect that new properties may be added and if not required, ignored. That is, clients should not break because they do not recognize additional properties.

Version Number Selection

Version number selection occurs after a client connection has been established with the service. By default, the service will for each client connection, use the lowest available major version of each message it supports.

The client is responsible for determining version compatibility. If compatible, the client must inform the service of its version requirements. If incompatible, the client must handle the incompatibilities, possibly by not using

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

incompatible commands. If the client cannot handle the incompatibilities then it should close the connection and not use the service.

The following sequence demonstrates use of the Common.Capabilities command to identify the command and event (both event and unsolicited) versions supported by the service, and the client use of the Common.SetVersions command to inform the service of the versions that should be used for the connection on which the command is sent.

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

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

Version Evolution Example

The following table depicts an example evolution of a command, an event and an unsolicited event.

Evolution	command	event	completion	unsolicited
Initial	1.0 propA	1.0 propA	1.0 propA	1.0 propA
Minor update - command property added - completion unchanged	**1.1** propA propB	1.0 propA	**1.1** propA	1.0 propA
Minor update - event property added	1.1 propA propB	**1.1** propA propB	1.1 propA	1.0 propA
Major update - completion property removed - unsolicited property removed - unsolicited property added	**2.0** propA propB	1.1 propA propB	**2.0** propB	**2.0** propB

Extending Enumeration Values

Extending an enumeration value is a breaking change as existing clients will not be coded to handle the new enumeration value. A breaking change to a message requires the message major version number be incremented.

Where possible the specification will avoid breaking changes. To support this, if the additional enumeration value is related to an existing enumeration value:

- An additional property with name *originalNameX* will be added to the message definition, where *originalName* is the original name of the property and *X* is the next available index. Indices will be non-negative integers and start at 2.
- The message minor version number will be incremented. This indicates the change is backwards compatible.
- The original property definition will be set as deprecated indicating it may be removed in a subsequent major revision of the message.
- Service implementations which implement the message version that defines the additional property will, if the original property is required, always include both *originalName* and *originalNameX* properties.

Existing clients will be unaffected by the additional property as the original property will still be included in the message. New or updated clients can be written to use any of the previous related properties. If a client does not have a use for a new enumeration value, it can continue to use one of the previously defined related properties.

For example, if version 1.0 of a message defines a *device* property with enumeration values:

- *online*, *offline*, *hardwareError*, *userError*

And a new enumeration value:

- *fraudAttempt*

Is added which relates to the existing, less specific, *userError* value, the new enumeration value could be added to the *device2* property in minor increment version 1.1 of the message. In this case when reporting the new enumeration value, version 1.1 of the message will include both:

```
{
  "device": "userError",
  "device2": "fraudAttempt"
}
```

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

1.2.4 End to End Security

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 device, 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 device. 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.

Details of end to end security are covered in the generic end to end security document (shared between XFS3.x and XFS4IoT.) Generic and specific E2E tokens are defined in that document. The tokens are passed to commands and returned in events which are documented in this specification, such as with CashDispenser.Dispense

There are specific commands to support E2E security which are covered by this document, including Common.GetCommandNonce and Common.ClearCommandNonce

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

2. Service Publisher

2.1 Summary

Multiple services can be supplied by multiple vendors. This standard doesn't require coordination between these different vendors, 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 applications, without the prior knowledge of any party.

This specification covers an environment using WebSockets to communicate between services and applications, 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.

2.2 General Information

2.2.1 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.

2.2.2 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 application and would need to be known by the organization setting up the application, but this configuration is outside the scope of this document.

2.2.3 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.

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

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 manufaturer 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.

2.2.4 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;

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

- 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.

2.2.5 Service Publishing

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

2.2.6 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

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.

2.2.7 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.*

2.2.8 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.

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

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).

2.2.9 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.

2.2.10 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.

2.2.11 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 "ServicePublisher.GetServices".

```
{
    "header": {
        "type": "command",
        "name": "ServicePublisher.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:

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

```
{
  "header": {
    "type": "completion",
    "name": "ServicePublisher.GetServicesCompletion",
    "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.*

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

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.

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

2.3 Command Messages

2.3.1 ServicePublisher.GetServices

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 canceled.
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	Yes
`}]`		
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

vendorName
Freeform string naming the hardware vendor

services/serviceURI
The URI which can be used to contact this individual service
Property value constraints:
`format: URI`

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

Event Messages

- ServicePublisher.ServiceDetailEvent

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

2.4 Event Messages

2.4.1 ServicePublisher.ServiceDetailEvent

Details of some services published by this endpoint

Payload	Type	Required
`{`		
`"vendorName": "ACME ATM Hardware GmbH",`	string	
`services": [{`	array (object)	
`"serviceURI": "wss://ATM1:123/xfs4iot/v1.0/CardReader"`	string	Yes
`}]`		
`}`		

Properties

vendorName
Freeform string naming the hardware vendor

services/serviceURI
The URI which can be used to contact this individual service
Property value constraints:
`format: URI`

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

3. Common Interface

This chapter defines the Common interface functionality and messages.

3.1 Summary

TODO

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

3.2 Command Messages

3.2.1 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 canceled.
default: 0

Completion Message

Payload	Type	Required
`{`		
`"common": {`	object	Yes
`"device": "online",`	string	
`"devicePosition": "inPosition",`	string	
`"powerSaveRecoveryTime": 0,`	integer	
`"antiFraudModule": "notSupported",`	string	
`"exchange": "notSupported"`	string	
`},`		
`"cardReader": {`	object	
`"media": "notSupported",`	string	
`"security": "notSupported",`	string	
`"chipPower": "notSupported",`	string	
`"chipModule": "ok",`	string	
`"magWriteModule": "ok",`	string	
`"frontImageModule": "ok",`	string	
`"backImageModule": "ok"`	string	
`},`		
`"cashAcceptor": {`	object	
`"intermediateStacker": "empty",`	string	

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

Payload	Type	Required
"stackerItems": "customerAccess",	string	
"banknoteReader": "ok",	string	
"dropBox": false,	boolean	
"positions": [{	array (object)	
"position": "inLeft",	string	
"shutter": "closed",	string	
"positionStatus": "empty",	string	
"transport": "ok",	string	
"transportStatus": "empty",	string	
"jammedShutterPosition": "notSupported"	string	
}],		
"mixedMode": "notActive"	string	
},		
"cashDispenser": {	object	
"intermediateStacker": "empty",	string	
"positions": [{	array (object)	
"position": "left",	string	
"shutter": "closed",	string	
"positionStatus": "empty",	string	
"transport": "ok",	string	
"transportStatus": "empty",	string	
"jammedShutterPosition": "notSupported"	string	
}]		
},		
"cashManagement": {	object	
"safeDoor": "doorNotSupported",	string	
"dispenser": "ok",	string	
"acceptor": "ok"	string	
},		
"keyManagement": {	object	
"encryptionState": "ready",	string	Yes
"certificateState": "unknown"	string	Yes
},		
"keyboard": {	object	
"autoBeepMode": "active"	string	Yes
},		
"textTerminal": {	object	

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

Payload	Type	Required
"keyboard": "on",	string	Yes
"keyLock": "on",	string	Yes
"displaySizeX": 0,	integer	Yes
"displaySizeY": 0	integer	Yes
},		
"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	
"*exampleProperty1*": "notSupported",	string	
"*exampleProperty2": "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	
"*exampleProperty1*": "unknown",	string	
"*exampleProperty2": "unknown"*	string	
},		
"blackMarkMode": "notSupported"	string	
},		

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

Payload	Type	Required
"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	Yes
"capture": false,	boolean	Yes
"dataPersistence": "persist",	string	Yes
"remainingStorage": 0	integer	Yes
},		
"camera": {	object	
"media": {	object	
"room": "ok",	string	
"person": "ok",	string	
"exitSlot": "ok",	string	
"*exampleProperty1*": "ok",	string	
"*exampleProperty2*": *"ok"*	string	
},		
"cameras": {	object	
"room": "notSupp",	string	
"person": "notSupp",	string	
"exitSlot": "notSupp",	string	
"*exampleProperty1*": "notSupp",	string	
"*exampleProperty2*": *"notSupp"*	string	
},		
"pictures": {	object	
"room": 0,	integer	
"person": 0,	integer	

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

Payload	Type	Required
"exitSlot": 0,	integer	
"*exampleProperty1*": 0,	integer	
"*exampleProperty2": 0*	integer	
}		
},		
"lights": {	object	
"cardReader": {	object	
"position": "left",	string	
"flashRate": "off",	string	
"colour": "red",	string	
"direction": "entry"	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	

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

Payload	Type	Required
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.		
},		
"statusGood": {	object	
See cardReader properties.		
},		
"statusWarning": {	object	
See cardReader properties.		
},		
"statusBad": {	object	
See cardReader properties.		
},		
"statusSupervisor": {	object	
See cardReader properties.		
},		
"statusInService": {	object	
See cardReader properties.		

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

Payload	Type	Required
`},`		
`"fasciaLight": {`	object	
`See cardReader properties.`		
`},`		
`"exampleProperty1": {`	object	
`See cardReader properties.`		
`},`		
`"exampleProperty2": {`	object	
`See exampleProperty1 properties.`		
`}`		
`},`		
`"auxiliaries": {`	object	
`"operatorSwitch": "notAvailable",`	string	
`"tamperSensor": "notAvailable",`	string	
`"intTamperSensor": "notAvailable",`	string	
`"seismicSensor": "notAvailable",`	string	
`"heatSensor": "notAvailable",`	string	
`"proximitySensor": "notAvailable",`	string	
`"ambientLightSensor": "notAvailable",`	string	
`"enhancedAudioSensor": "notAvailable",`	string	
`"bootSwitchSensor": "notAvailable",`	string	
`"displaySensor": "notAvailable",`	string	
`"operatorCallButtonSensor": "notAvailable",`	string	
`"handsetSensor": "notAvailable",`	string	
`"headsetMicrophoneSensor": "notAvailable",`	string	
`"FasciaMicrophoneSensor": "notAvailable",`	string	
`"safeDoor": "notAvailable",`	string	
`"vandalShield": "notAvailable",`	string	
`"cabinetFrontDoor": "notAvailable",`	string	
`"cabinetRearDoor": "notAvailable",`	string	
`"cabinetLeftDoor": "notAvailable",`	string	
`"cabinetRightDoor": "notAvailable",`	string	
`"openClosedIndicator": "notAvailable",`	string	
`"audio": {`	object	
`"notAvailable": false,`	boolean	
`"off": false,`	boolean	
`"keypress": false,`	boolean	

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

Payload	Type	Required
"exclamation": false,	boolean	
"warning": false,	boolean	
"error": false,	boolean	
"critical": false,	boolean	
"continuous": false	boolean	
},		
"heating": "notAvailable",	string	
"consumerDisplayBacklight": "notAvailable",	string	
"signageDisplay": "notAvailable",	string	
volume": {	object	
"available": false,	boolean	
"volumeLevel": 1	integer	
},		
"UPS": {	object	
"notAvailable": false,	boolean	
"low": false,	boolean	
"engaged": false,	boolean	
"powering": false,	boolean	
"recovered": false	boolean	
},		
"audibleAlarm": "notAvailable",	string	
"enhancedAudioControl": "notAvailable",	string	
"enhancedMicrophoneControl": "notAvailable",	string	
microphoneVolume": {	object	
"available": false,	boolean	
"volumeLevel": 1	integer	
}		
},		
"vendorMode": {	object	
"device": "online",	string	
"service": "enterPending"	string	
},		
"vendorApplication": {	object	
"accessLevel": "notActive"	string	
}		
}		

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

Properties
common Status information common to all XFS4IoT services.
common/device Specifies the state of the device. Following values are possible: • `online` - The device is online. This is returned when the device is present and operational. • `offline` - The device is offline (e.g. the operator has taken the device offline by turning a switch). • `powerOff` - The device is powered off or physically not connected. • `noDevice` - The device is not intended to be there, e.g. this type of self service machine does not contain such a device or it is internally not configured. • `hardwareError` - The device is inoperable due to a hardware error. • `userError` - The device is present but a person is preventing proper device operation. • `deviceBusy` - The device is busy and unable to process a command at this time. • `fraudAttempt` - The device is present but is inoperable because it has detected a fraud attempt. • `potentialFraud` - The device has detected a potential fraud attempt and is capable of remaining in service. In this case the application should make the decision as to whether to take the device offline.
common/devicePosition Position of the device. Following values are possible: • `inPosition` - The device is in its normal operating position, or is fixed in place and cannot be moved. • `notInPosition` - The device has been removed from its normal operating position. • `unknown` - Due to a hardware error or other condition, the position of the device cannot be determined.
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. Following values are possible: • `notSupported` - No anti-fraud module is available. • `ok` - Anti-fraud module is in a good state and no foreign device is detected. • `inoperable` - Anti-fraud module is inoperable. • `deviceDetected` - Anti-fraud module detected the presence of a foreign device. • `unknown` - The state of the anti-fraud module cannot be determined.
common/exchange Specifies the exchange state of the service. Exchange can used to perform a manual replenishment of a device and is entered by Storage.StartExchange and completed by Storage.EndExchange. When the state changes a Common.ExchangeStateChangedEvent is posted. Following values are possible: • `notSupported` - Exchange is not supported on this service. • `active` - Exchange is active on this service. Commands which interact with the device may be rejected with an error code as appropriate. • `inactive` - Exchange is not active on this service. default: "notSupported"
cardReader Status information for XFS4IoT services implementing the CardReader interface. This will be omitted if the CardReader interface is not supported.

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

Properties
cardReader/media Specifies the transport/exit position media state as one of the following values: • `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. 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/security Specifies the state of the security module 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/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.

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

Properties
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.
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 application 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. If true, some items are stored in the drop box due to a cash-in transaction which caused a problem. If false, the drop box is empty or there is no drop box.

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

Properties
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 *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.

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

Properties
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.

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

Properties
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. • `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. • `notEmptyUnkown` - 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.

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

Properties
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. • `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 storage units for dispensing cash. Following values are possible: • `ok` - All storage units present are in a good state. • `attention` - One or more of the storage units is in a low, empty, inoperative or manipulated condition. Items can still be dispensed from at least one of the storage units. • `stop` - Due to a storage unit failure dispensing is impossible. No items can be dispensed because all of the storage units are in an empty, inoperative or manipulated condition. This state may also occur when a reject/retract storage unit is full or no reject/retract storage unit is present, or when an application lock is set on every storage unit which can be locked. • `unknown` - Due to a hardware error or other condition, the state of the storage units cannot be determined.
cashManagement/acceptor Supplies the state of the storage units for accepting cash. Following values are possible: • `ok` - All storage units present are in a good state. • `attention` - One or more of the storage units is in a high, full, inoperative or manipulated condition. Items can still be accepted into at least one of the storage units. • `stop` - Due to a storage unit failure accepting is impossible. No items can be accepted because all of the storage units are in a full, inoperative or manipulated condition. This state may also occur when a retract storage unit is full or no retract cash storage unit is present, or when an application lock is set on every storage unit, or when Level 2/3 notes are to be automatically retained within storage units, but all of the designated storage units for storing them are full or inoperative. • `unknown` - Due to a hardware error or other condition, the state of the storage 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 Specifies the state of the encryption module.

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

Properties
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). The following values are possible: • `active` - An automatic tone will be generated for all active keys. • `inActive` - An automatic tone will be generated for all 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 in the text terminal unit as one of the following values: • `on` - The keyboard is activated. • `off` - The keyboard is not activated. • `notAvailable` - The keyboard is not available.
textTerminal/keyLock Specifies the state of the keyboard lock of the text terminal unit as one of the following values: • `on` - The keyboard lock switch is activated. • `off` - The keyboard lock switch is not activated. • `notAvailable` - The keyboard lock switch is not available.
textTerminal/displaySizeX Specifies the horizontal size of the display of the text terminal unit. Property value constraints: `minimum: 0`
textTerminal/displaySizeY Specifies the vertical size of the display of the text terminal unit. Property value constraints: `minimum: 0`
printer Status information for XFS4IoT services implementing the Printer interface. This will be omitted if the Printer interface is not supported.

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

Properties
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.

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

Properties
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. • `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.

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

Properties
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.

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

Properties
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.

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

Properties
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. The following values are possible: • `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. • `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.
camera Status information for XFS4IoT services implementing the Camera interface. This will be omitted if the Camera interface is not supported.
camera/media Specifies the state of the recording media of the cameras as one of the following. For a device which stores pictures on a hard disk drive or other general-purpose storage, the value of the media field should be notSupp. • `ok` - The media is in a good state. • `high` - The media is almost full (threshold). • `full` - The media is full. • `notSupp` - The device does not support sensing the media level. • `unknown` - Due to a hardware error or other condition, the state of the media cannot be determined.
camera/media/room Specifies the state of the recording media of the camera that monitors the whole self-service area.
camera/media/person Specifies the state of the recording media of the camera that monitors the person standing in front of the self-service machine.
camera/media/exitSlot Specifies the state of the recording media of the camera that monitors the exit slot(s) of the self-service machine.
camera/cameras Specifies the state of the cameras as one of the following. • `notSupp` - The camera is not supported. • `ok` - The camera is in a good state. • `inop` - The camera is inoperative. • `unknown` - Due to a hardware error or other condition, the state of the camera cannot be determined.
camera/cameras/room Specifies the state of the camera that monitors the whole self-service area.
camera/cameras/person Specifies the state of the camera that monitors the person standing in front of the self-service machine.
camera/cameras/exitSlot Specifies the state of the camera that monitors the exit slot(s) of the self-service machine.

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

Properties
camera/pictures Specifies the number of pictures stored on the recording media of the cameras. For a device which stores pictures on a hard disk drive or other general-purpose storage, the value of the pictures field should be zero.
camera/pictures/room Specifies the number of pictures stored on the recording media of the room camera Property value constraints: `minimum: 0`
camera/pictures/person Specifies the number of pictures stored on the recording media of the person camera Property value constraints: `minimum: 0`
camera/pictures/exitSlot Specifies the number of pictures stored on the recording media of the exit slot camera Property value constraints: `minimum: 0`
camera/pictures/exampleProperty1 (example name) Property value constraints: `minimum: 0`
lights Status information for XFS4IoT services implementing the Lights interface. This will be omitted if the Lights interface is not supported.
lights/cardReader Card Unit Light.
lights/cardReader/position The light position. If omitted then the default position is used. One of the following values: • `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.
lights/cardReader/flashRate The light flash rate as one of the following values: • `off` - The light is turned off. • `slow` - The light is flashing slowly. • `medium` - The light is flashing medium frequency. • `quick` - The light is flashing quickly. • `continuous` - The light is continuous (steady).

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

Properties
lights/cardReader/colour The light color as one of the following values: • `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.
lights/cardReader/direction The light direction as one of the following values: • `entry` - The light is indicating entry. • `exit` - The light is indicating exit.
lights/pinPad Pin Pad Light.
lights/notesDispenser Notes Dispenser Light.
lights/coinDispenser Coin Dispenser Light.
lights/receiptPrinter Receipt Printer Light.
lights/passbookPrinter Passbook Printer Light.
lights/envelopeDepository Envelope Depository Light.
lights/chequeUnit Cheque Unit Light.
lights/billAcceptor Bill Acceptor Light.
lights/envelopeDispenser Envelope Dispenser Light.
lights/documentPrinter Document Printer Light.
lights/coinAcceptor Coin Acceptor Light.
lights/scanner scanner Light.
lights/contactless Contactless Light.
lights/cardUnit2 Card Unit 2 Light.

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

Properties
lights/notesDispenser2 Notes Dispenser 2 Light.
lights/billAcceptor2 Bill Acceptor 2 Light.
lights/statusGood Status indicator light - Good.
lights/statusWarning Status indicator light - Warning.
lights/statusBad Status indicator light - Bad.
lights/statusSupervisor Status indicator light - Supervisor.
lights/statusInService Status indicator light - In Service.
lights/fasciaLight Fascia Light.
lights/exampleProperty1 (example name)
auxiliaries Status information for XFS4IoT services implementing the Auxiliaries interface. This will be omitted if the Auxiliaries interface is not supported.
auxiliaries/operatorSwitch Specifies the state of the Operator switch. • `notAvailable` - The status is not available. • `run` - The switch is in run mode. • `maintenance` - The switch is in maintenance mode. • `supervisor` - TThe switch is in supervisor mode.
auxiliaries/tamperSensor Specifies the state of the Tamper sensor. • `notAvailable` - The tamper sensor is not available. • `off` - There is no indication of a tampering attempt. • `on` - There has been a tampering attempt.
auxiliaries/intTamperSensor Specifies the state of the Internal Tamper Sensor for the internal alarm. This sensor indicates whether the internal alarm has been tampered with (such as a burglar attempt). Specified as one of the following: • `notAvailable` - The internal tamper sensor is not available. • `off` - There is no indication of a tampering attempt. • `on` - There has been a tampering attempt.
auxiliaries/seismicSensor Specifies the state of the Seismic Sensor. This sensor indicates whether the terminal has been shaken (e.g. burglar attempt or seismic activity). Specified as one of the following: • `notAvailable` - The status sensor is not available. • `off` - TThe seismic activity has not been high enough to trigger the sensor. • `on` - The seismic or other activity has triggered the sensor.

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

Properties
auxiliaries/heatSensor Specifies the state of the Heat Sensor. This sensor is triggered by excessive heat (fire) near the terminal. Specified as one of the following: • `notAvailable` - The status is not available. • `off` - The heat has not been high enough to trigger the sensor. • `on` - The heat has been high enough to trigger the sensor.
auxiliaries/proximitySensor Specifies the state of the Proximity Sensor. This sensor is triggered by movements around the terminal. Specified as one of the following: • `notAvailable` - The status is not available. • `present` - The sensor is showing that there is someone present at the terminal. • `notPresent` - The sensor can not sense any people around the terminal.
auxiliaries/ambientLightSensor Specifies the state of the Ambient Light Sensor. This sensor indicates the level of ambient light around the terminal. Interpretation of this value is vendor-specific and therefore it is not guaranteed to report a consistent actual ambient light level across different vendor hardware. Specified as one of the following: • `notAvailable` - The status is not available. • `veryDark` - The level of light is very dark. • `dark` - The level of light is dark. • `mediumLight` - The level of light is medium light. • `light` - The level of light is light. • `veryLight` - The level of light is very light.
auxiliaries/enhancedAudioSensor Specifies the presence or absence of a consumer's headphone connected to the Audio Jack. Specified as one of the following: • `notAvailable` - The status is not available. • `present` - There is a headset connected. • `notPresent` - There is no headset connected.
auxiliaries/bootSwitchSensor Specifies the state of the Boot Switch Sensor. This sensor is triggered whenever the terminal is about to be rebooted or shutdown due to a delayed effect switch. Specified as one of the following: • `notAvailable` - The status is not available. • `off` - The sensor has not been triggered. • `on` - The terminal is about to be rebooted or shutdown.
auxiliaries/displaySensor Specifies the state of the Consumer Display. Specified as one of the following: • `notAvailable` - The status is not available. • `off` - The Consumer Display is switched off. • `on` - The Consumer Display is in a good state and is turned on. • `displayError` - The Consumer Display is in an error state.
auxiliaries/operatorCallButtonSensor Specifies the state of the Operator Call Button as one of the following: • `notAvailable` - The status is not available. • `off` - The Operator Call Button is released (not pressed). • `on` - The Operator Call Button is being pressed.

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

Properties
auxiliaries/handsetSensor Specifies the state of the Handset, which is a device similar to a telephone receiver. Specified as one of the following: • `notAvailable` - The status is not available. • `onTheHook` - The Handset is on the hook. • `offTheHook` - The Handset is off the hook.
auxiliaries/headsetMicrophoneSensor Specifies the presence or absence of a consumer's headset microphone connected to the Microphone Jack. Specified as one of the following: • `notAvailable` - The status is not available. • `present` - There is a headset microphone connected. • `notPresent` - There is no headset microphone connected.
auxiliaries/FasciaMicrophoneSensor Specifies the state of the fascia microphone as one of the following: • `notAvailable` - The status is not available. • `off` - The Fascia Microphone is turned off. • `on` - The Fascia Microphone is turned on.
auxiliaries/safeDoor Specifies the state of the Safe Doors. Safe Doors are doors that open up for secure hardware, such as the note dispenser, the security device, etc. Specified as one of the following: • `notAvailable` - The status is not available. • `closed` - The Safe Doors are closed. • `open` - At least one of the Safe Doors is open. • `locked` - All Safe Doors are closed and locked. • `bolted` - All Safe Doors are closed, locked and bolted. • `tampered` - At least one of the Safe Doors has potentially been tampered with.
auxiliaries/vandalShield Specifies the state of the Vandal Shield. The Vandal Shield is a door that opens up for consumer access to the terminal. Specified as one of the following: • `notAvailable` - The status is not available. • `closed` - The Vandal Shield is closed. • `open` - The Vandal Shield is open. • `locked` - The Vandal Shield is closed and locked. • `service` - The Vandal Shield is in service position. • `keyboard` - The Vandal Shield position permits access to the keyboard. • `ajar` - The Vandal Shield is ajar. • `jammed` - The Vandal Shield is jammed. • `tampered` - The Vandal Shield has potentially been tampered with.

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

Properties
auxiliaries/cabinetFrontDoor Specifies the overall state of the Front Cabinet Doors. The front is defined as the side facing the customer/consumer. Cabinet Doors are doors that open up for consumables, and hardware that does not have to be in a secure place. Specified as one of the following: • `notAvailable` - The status is not available. • `closed` - All front Cabinet Doors are closed. • `open` - At least one of the front Cabinet Doors is open. • `locked` - All front Cabinet Doors are closed and locked. • `bolted` - All front Cabinet Doors are closed, locked and bolted. • `tampered` - At least one of the front Cabinet Doors has potentially been tampered with.
auxiliaries/cabinetRearDoor Specifies the overall state of the Rear Cabinet Doors. The front is defined as the side opposite the side facing the customer/consumer. Cabinet Doors are doors that open up for consumables, and hardware that does not have to be in a secure place. Specified as one of the following: • `notAvailable` - The status is not available. • `closed` - All rear Cabinet Doors are closed. • `open` - At least one of the rear Cabinet Doors is open. • `locked` - All rear Cabinet Doors are closed and locked. • `bolted` - All rear Cabinet Doors are closed, locked and bolted. • `tampered` - At least one of the rear Cabinet Doors has potentially been tampered with.
auxiliaries/cabinetLeftDoor Specifies the overall state of the Left Cabinet Doors. The left is defined as the side to the left as seen by the customer/consumer. Cabinet Doors are doors that open up for consumables, and hardware that does not have to be in a secure place. Specified as one of the following: • `notAvailable` - The status is not available. • `closed` - All left Cabinet Doors are closed. • `open` - At least one of the left Cabinet Doors is open. • `locked` - All left Cabinet Doors are closed and locked. • `bolted` - All left Cabinet Doors are closed, locked and bolted. • `tampered` - At least one of the left Cabinet Doors has potentially been tampered with.
auxiliaries/cabinetRightDoor Specifies the overall state of the Right Cabinet Doors. The right is defined as the side to the right as seen by the customer/consumer. Cabinet Doors are doors that open up for consumables, and hardware that does not have to be in a secure place. Specified as one of the following: • `notAvailable` - The status is not available. • `closed` - All right Cabinet Doors are closed. • `open` - At least one of the right Cabinet Doors is open. • `locked` - All right Cabinet Doors are closed and locked. • `bolted` - All right Cabinet Doors are closed, locked and bolted. • `tampered` - At least one of the right Cabinet Doors has potentially been tampered with.
auxiliaries/openClosedIndicator Specifies the state of the Open/Closed Indicator as one of the following: • `notAvailable` - The status is not available. • `closed` - The terminal is closed for a consumer. • `open` - The terminal is open to be used by a consumer.

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

Properties
auxiliaries/audio Specifies the state of the Audio Indicator as either one or a combination of the following values. Interpretation of this value is vendor-specific and therefore it is not possible to guarantee a consistent actual sound pattern across different vendor hardware: • `notAvailable` - The status is not available. • `off` - The Audio Indicator is turned off. • `keypress` - The Audio Indicator sounds a key click signal. • `exclamation` - The Audio Indicator sounds an exclamation signal. • `warning` - The Audio Indicator sounds a warning signal. • `error` - The Audio Indicator sounds an error signal. • `critical` - The Audio Indicator sounds a critical signal. • `continuous` - The Audio Indicator is turned on continuously.
auxiliaries/audio/notAvailable default: false
auxiliaries/audio/off default: false
auxiliaries/audio/keypress default: false
auxiliaries/audio/exclamation default: false
auxiliaries/audio/warning default: false
auxiliaries/audio/error default: false
auxiliaries/audio/critical default: false
auxiliaries/audio/continuous default: false
auxiliaries/heating Specifies the state of the internal heating as one of the following: • `notAvailable` - The status is not available. • `off` - The internal heating is turned off. • `on` - The internal heating is turned on.
auxiliaries/consumerDisplayBacklight Specifies the Consumer Display Backlight as one of the following: • `notAvailable` - The status is not available. • `off` - The Consumer Display Backlight is turned off. • `on` - Consumer Display Backlight is turned on.

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

Properties
auxiliaries/signageDisplay Specifies the state of the Signage Display. The Signage Display is a lighted banner or marquee that can be used to display information or an advertisement. Any dynamic data displayed must be loaded by a means external to the Service Provider. Specified as one of the following: • `notAvailable` - The status is not available. • `off` - The Signage Display is turned off. • `on` - The Signage Display is turned on.
auxiliaries/volume/available Specifies if the volume control is available. Possible values: • `false` - The volume control is not available. • `true` - The volume control is available. default: false

auxiliaries/volume/volumeLevel
Specifies the value of the Volume Control, if available. 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. The interval is defined in logarithmic steps, e.g. a volume control on a radio. Note: The Volume Control property is vendor-specific and therefore it is not possible to guarantee a consistent actual volume level across different vendor hardware.
Property value constraints:
`minimum: 1`
`maximum: 1000`

auxiliaries/UPS Specifies the state of the Uninterruptible Power Supply device as a combination of the following values: • `notAvailable` - The status is not available. • `low` - The charge level of the UPS is low. • `engaged` - TThe UPS is engaged. • `powering` - The UPS is powering the system. • `recovered` - The UPS was engaged when the main power went off.
auxiliaries/UPS/notAvailable default: false
auxiliaries/UPS/low default: false
auxiliaries/UPS/engaged default: false
auxiliaries/UPS/powering default: false
auxiliaries/UPS/recovered default: false
auxiliaries/audibleAlarm Species the state of the Audible Alarm device as one of the following: • `notAvailable` - The status is not available. • `off` - The Alarm is turned off. • `on` - The Alarm is turned on.

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

Properties
auxiliaries/enhancedAudioControl Specifies the state of the Enhanced Audio Controller. The Enhanced Audio Controller controls how private and public audio are broadcast when the headset is inserted into/removed from the audio jack and when the handset is off-hook/on-hook. In the following, Privacy Device is used to refer to either the headset or handset. The Enhanced Audio Controller state is specified as one of the following: • `notAvailable` - The status is not available. • `publicAudioManual` - The Enhanced Audio Controller is in manual mode and is in the public state (i.e. audio will be played through speakers). Activating a Privacy Device (headset connected/handset off-hook) will have no impact, i.e. Output will remain through the speakers & no audio will be directed to the Privacy Device. • `publicAudioAuto` - The Enhanced Audio Controller is in auto mode and is in the public state (i.e. audio will be played through speakers). When a Privacy Device is activated, the device will go to the private state. • `publicAudioSemiAuto` - The Enhanced Audio Controller is in semi-auto mode and is in the public state (i.e. audio will be played through speakers). When a Privacy Device is activated, the device will go to the private state. • `privateAudioManual` - he Enhanced Audio Controller is in manual mode and is in the private state (i.e. audio will be played only through a connected Privacy Device). In private mode, no audio is transmitted through the speakers. • `privateAudioAuto` - The Enhanced Audio Controller is in auto mode and is in the private state (i.e. audio will be played only through a connected Privacy Device). In private mode, no audio is transmitted through the speakers. When a Privacy Device is deactivated (headset disconnected/handset on-hook), the device will go to the public state. Where there is more than one Privacy Device, the device will go to the public state only when all Privacy Devices have been deactivated. • `privateAudioSemiAuto` - The Enhanced Audio Controller is in semi-auto mode and is in the private state (i.e. audio will be played only through a connected Privacy Device). In private mode, no audio is transmitted through the speakers. When a Privacy Device is deactivated, the device will remain in the private state.

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

Properties
auxiliaries/enhancedMicrophoneControl Specifies the state of the Enhanced Microphone Controller. The Enhanced Microphone Controller controls how private and public audio input are transmitted when the headset is inserted into/removed from the audio jack and when the handset is off-hook/on-hook. In the following, Privacy Device is used to refer to either the headset or handset. The Enhanced Microphone Controller state is specified as one of the followings: • `notAvailable` - The status is not available. • `publicAudioManual` - The Enhanced Microphone Controller is in manual mode and is in the public state (i.e. the microphone in the fascia is active). Activating a Privacy Device (headset connected/handset off-hook) will have no impact, i.e. input will remain through the fascia microphone and any microphone associated with the Privacy Device will not be active • `publicAudioAuto` - The Enhanced Microphone Controller is in auto mode and is in the public state (i.e. the microphone in the fascia is active). When a Privacy Device with a microphone is activated, the device will go to the private state. • `publicAudioSemiAuto` - The Enhanced Microphone Controller is in semi-auto mode and is in the public state (i.e. the microphone in the fascia is active). When a Privacy Device with a microphone is activated, the device will go to the private state. • `privateAudioManual` - The Enhanced Microphone Controller is in manual mode and is in the private state (i.e. audio input will be via a microphone in the Privacy Device). In private mode, no audio input is transmitted through the fascia microphone. • `privateAudioAuto` - The Enhanced Microphone Controller is in auto mode and is in the private state (i.e. audio input will be via a microphone in the Privacy Device). In private mode, no audio input is transmitted through the fascia microphone. When a Privacy Device with a microphone is deactivated (headset disconnected/handset on-hook), the device will go to the public state. Where there is more than one Privacy Device with a microphone, the device will go to the public state only when all such Privacy Devices have been deactivated. • `privateAudioSemiAuto` - The Enhanced Microphone Controller is in semi-auto mode and is in the private state (i.e. audio input will be via a microphone in the Privacy Device). In private mode, no audio is transmitted through the fascia microphone. When a Privacy Device with a microphone is deactivated, the device will remain in the private state.
auxiliaries/microphoneVolume/available Specifies if the Microphone Volume Control is available. Possible values: • `false` - The volume control is not available. • `true` - The volume control is available. default: false

auxiliaries/microphoneVolume/volumeLevel
Specifies the value of the Microphone Volume Control, if available. 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. The interval is defined in logarithmic steps, e.g. a volume control on a radio. Note: The Microphone Volume Control property is vendor-specific and therefore it is not possible to guarantee a consistent actual volume level across different vendor hardware.
Property value constraints:
`minimum: 1`
`maximum: 1000`

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

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

Properties
vendorMode/device Specifies the status of the Vendor Mode Service Provider. Status will be one of the following values: • `online` - The Vendor Mode service is available. • `offline` - The Vendor Mode service is not available.
vendorMode/service Specifies the service state as one of the following values: • `enterPending` - Vendor Mode enter request pending. • `active` - Vendor Mode active. • `exitPending` - Vendor Mode exit request pending. • `inactive` - Vendor Mode inactive.
vendorApplication Status information for XFS4IoT services implementing the Vendor Application interface. This will be omitted if the Vendor Mode interface is not supported.
vendorApplication/accessLevel Reports the current access level as one of the following values: • `notActive` - The application is not active. • `basic` - The application is active for the basic access level. • `intermediate` - The application is active for the intermediate access level. • `full` - The application is active for the full access level.

Event Messages

None

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

3.2.2 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 canceled. default: 0

Completion Message

Payload	Type	Required
`{`		
`  "interfaces": [{`	array (object)	Yes
`    "name": "Common",`	string	
`    "commands": {`	object	
`      "CardReader.ReadRawData": {`	object	
`          "versions": ["1.3", "2.1", "3.0"]`	array (string)	
`      },`		
`      "CardReader.EjectCard": {`	object	
`        See CardReader.ReadRawData properties.`		
`      }`		
`    },`		
`    "events": {`	object	
`      "CardReader.MediaInsertedEvent": {`	object	
`          "versions": ["1.3", "2.1", "3.0"]`	array (string)	
`      },`		
`      "CardReader.MediaTakenEvent": {`	object	
`        See CardReader.MediaInsertedEvent properties.`		
`      }`		
`    },`		
`    "maximumRequests": 0,`	integer	
`    "authenticationRequired": [Add example to YAML]`	array (string)	
`  }],`		

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

Payload	Type	Required
"common": {	object	Yes
"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)	
"softwareName": Add example to YAML,	string	
"softwareVersion": Add example to YAML	string	
}]		
}],		
"vendorModeIformation": {	object	
"allowOpenSessions": false,	boolean	
"allowedExecuteCommands": [Add example to YAML]	array (string)	
},		
"powerSaveControl": false,	boolean	
"antiFraudModule": false,	boolean	
"synchronizableCommands": [Add example to YAML],	array (string)	
"endToEndSecurity": true,	boolean	
"hardwareSecurityElement": true,	boolean	
"responseSecurityEnabled": true,	boolean	
"commandNonceTimeout": 3600	integer	
},		
"cardReader": {	object	
"type": "motor",	string	
"readTracks": {	object	
"track1": false,	boolean	
"track2": false,	boolean	
"track3": false,	boolean	
"watermark": false,	boolean	
"frontTrack1": false,	boolean	

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

Payload	Type	Required
"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	
},		
"securityType": "notSupported",	string	
"powerOnOption": "notSupported",	string	
"powerOffOption": "notSupported",	string	
"fluxSensorProgrammable": false,	boolean	
"readWriteAccessFromExit": false,	boolean	
"writeMode": {	object	
"notSupported": false,	boolean	
"loco": false,	boolean	
"hico": false,	boolean	
"auto": false	boolean	
},		
"chipPower": {	object	
"notSupported": false,	boolean	
"cold": false,	boolean	

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

Payload	Type	Required
"warm": false,	boolean	
"off": false	boolean	
},		
"memoryChipProtocols": {	object	
"siemens4442": false,	boolean	
"gpm896": false	boolean	
},		
"positions": {	object	
"exit": false,	boolean	
"transport": false	boolean	
},		
"cardTakenSensor":	undefined	
},		
"cashAcceptor": {	object	
"type": "tellerBill",	string	
"maxCashInItems": 1,	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	
},		

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

Payload	Type	Required
"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	
},		
"countActions": {	object	
"individual": false,	boolean	
"all": false	boolean	
},		
"deviceLockControl": false,	boolean	
"deplete": false,	boolean	
"counterfeitAction": "none"	string	
},		
"cashDispenser": {	object	

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

Payload	Type	Required
"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	

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

Payload	Type	Required
"toTransport": false,	boolean	
"toStacker": false	boolean	
},		
"prepareDispense": false	boolean	
},		
"cashManagement": {	object	
"safeDoor": false,	boolean	
"cashBox": false,	boolean	
"exchangeType": {	object	
"byHand": false,	boolean	
"toCassettes": false,	boolean	
"clearRecycler": false,	boolean	
"depositInto": false	boolean	
},		
"itemInfoTypes": {	object	
"serialNumber": false,	boolean	
"signature": false,	boolean	
"imageFile": false	boolean	
},		
"classificationList": false,	boolean	
"physicalNoteList": false	boolean	
},		
"pinPad": {	object	
"pinFormats": {	object	Yes
"ibm3624": false,	boolean	
"ansi": false,	boolean	
"iso0": false,	boolean	
"iso1": false,	boolean	
"eci2": false,	boolean	
"eci3": false,	boolean	
"visa": false,	boolean	
"diebold": false,	boolean	
"dieboldCo": false,	boolean	
"visa3": false,	boolean	
"emv": false,	boolean	
"iso3": false,	boolean	
"ap": false	boolean	

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

Payload	Type	Required
`},`		
`"presentationAlgorithms": {`	object	Yes
`"presentClear": false`	boolean	
`},`		
`"display": {`	object	Yes
`"none": false,`	boolean	
`"ledThrough": false,`	boolean	
`"display": false`	boolean	
`},`		
`"idcConnect": false,`	boolean	Yes
`"validationAlgorithms": {`	object	Yes
`"des": false,`	boolean	
`"visa": false`	boolean	
`},`		
`"pinCanPersistAfterUse": false,`	boolean	Yes
`"typeCombined": false,`	boolean	Yes
`"setPinblockDataRequired": false,`	boolean	Yes
`"pinBlockAttributes": {`	object	Yes
`"P0": {`	object	
`"T": {`	object	
`"E": {`	object	
`"cryptoMethod": {`	object	Yes
`"ecb": false,`	boolean	
`"cbc": false,`	boolean	
`"cfb": false,`	boolean	
`"ofb": false,`	boolean	
`"ctr": false,`	boolean	
`"xts": false,`	boolean	
`"rsaesPkcs1V15": false,`	boolean	
`"rsaesOaep": false`	boolean	
`}`		
`}`		
`},`		
`"R": {`	object	
See T properties.		
`}`		
`}`		

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

Payload	Type	Required
`}`		
`},`		
`"crypto": {`	object	
`"emvHashAlgorithm": {`	object	
`"sha1Digest": false,`	boolean	
`"sha256Digest": false`	boolean	
`},`		
`"cryptoAttributes": {`	object	
`"D0": {`	object	
`"D": {`	object	
`"D": {`	object	
`"cryptoMethod": {`	object	Yes
`"ecb": false,`	boolean	
`"cbc": false,`	boolean	
`"cfb": false,`	boolean	
`"ofb": false,`	boolean	
`"ctr": false,`	boolean	
`"xts": false,`	boolean	
`"rsaesPkcs1V15": false,`	boolean	
`"rsaesOaep": false`	boolean	
`}`		
`},`		
`"E": {`	object	
`See D properties.`		
`}`		
`},`		
`"T": {`	object	
`See D properties.`		
`}`		
`},`		
`"D1": {`	object	
`See D0 properties.`		
`}`		
`},`		
`"authenticationAttributes": {`	object	
`"S0": {`	object	
`"R": {`	object	

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

Payload	Type	Required
"G": {	object	
"cryptoMethod": {	object	
"rsassaPkcs1V15": false,	boolean	
"rsassaPss": false	boolean	
},		
"hashAlgorithm": {	object	
"sha1": false,	boolean	
"sha256": false	boolean	
}		
},		
"S": {	object	
See G properties.		
}		
},		
"T": {	object	
See R properties.		
}		
},		
"M0": {	object	
See S0 properties.		
}		
},		
"verifyAttributes": {	object	
"S0": {	object	
"R": {	object	
"V": {	object	
"cryptoMethod": {	object	
"rsassaPkcs1V15": false,	boolean	
"rsassaPss": false	boolean	
},		
"hashAlgorithm": {	object	
"sha1": false,	boolean	
"sha256": false	boolean	
}		
}		
},		
"T": {	object	

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

Payload	Type	Required
See R properties.		
}		
},		
"M0": {	object	
See S0 properties.		
}		
}		
},		
"keyManagement": {	object	
"keyNum": 0,	integer	Yes
"derivationAlgorithms": {	object	
"chipZka": false	boolean	
},		
"keyCheckModes": {	object	Yes
"self": false,	boolean	
"zero": false	boolean	
},		
"hsmVendor": Add example to YAML,	string	
"rsaAuthenticationScheme": {	object	Yes
"2partySig": false,	boolean	
"3partyCert": false,	boolean	
"3partyCertTr34": false	boolean	
},		
"rsaSignatureAlgorithm": {	object	Yes
"pkcs1V15": false,	boolean	
"pss": false	boolean	
},		
"rsaCryptAlgorithm": {	object	Yes
"pkcs1V15": false,	boolean	
"oaep": false	boolean	
},		
"rsaKeyCheckMode": {	object	Yes
"sha1": false,	boolean	
"sha256": false	boolean	
},		
"signatureScheme": {	object	Yes
"genRsaKeyPair": false,	boolean	

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

Payload	Type	Required
"randomNumber": false,	boolean	
"exportEppId": false,	boolean	
"enhancedRkl": false	boolean	
},		
"emvImportSchemes": {	object	Yes
"plainCA": false,	boolean	
"chksumCA": false,	boolean	
"epiCA": false,	boolean	
"issuer": false,	boolean	
"icc": false,	boolean	
"iccPin": false,	boolean	
"pkcsv15CA": false	boolean	
},		
"keyBlockImportFormats": {	object	Yes
"ansTr31KeyBlock": false,	boolean	
"ansTr31KeyBlockB": false,	boolean	
"ansTr31KeyBlockC": false	boolean	
},		
"keyImportThroughParts": false,	boolean	Yes
"desKeyLength": {	object	Yes
"single": false,	boolean	
"double": false,	boolean	
"triple": false	boolean	
},		
"certificateTypes": {	object	Yes
"encKey": false,	boolean	
"verificationKey": false,	boolean	
"hostKey": false	boolean	
},		
"loadCertOptions": [{	array (object)	Yes
"signer": "certHost",	string	
"option": {	object	
"newHost": false,	boolean	
"replaceHost": false	boolean	
}		
}],		
"crklLoadOptions": {	object	Yes

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

Payload	Type	Required
"noRandom": false,	boolean	
"noRandomCrl": false,	boolean	
"random": false,	boolean	
"randomCrl": false	boolean	
},		
"symmetricKeyManagementMethods": {	object	Yes
"fixedKey": false,	boolean	
"masterKey": false,	boolean	
"tdesDukpt": false	boolean	
},		
"keyAttributes": {	object	Yes
"*M0*": {	object	
"*T*": {	object	
"*C*": {	object	
"restricted": Add example to YAML	string	
},		
"*E*": {	object	
See C properties.		
}		
},		
"*R*": {	object	
See T properties.		
}		
},		
"*K1*": {	object	
See M0 properties.		
}		
},		
"decryptAttributes": {	object	Yes
"*A*": {	object	
"decryptMethod": {	object	Yes
"ecb": false,	boolean	
"cbc": false,	boolean	
"cfb": false,	boolean	
"ofb": false,	boolean	
"ctr": false,	boolean	
"xts": false,	boolean	

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

Payload	Type	Required
"rsaesPkcs1V15": false,	boolean	
"rsaesOaep": false	boolean	
}		
},		
"T": {	object	
See A properties.		
}		
},		
"verifyAttributes": {	object	Yes
"M0": {	object	
"T": {	object	
"V": {	object	
"cryptoMethod": {	object	Yes
"kcvNone": false,	boolean	
"kcvSelf": false,	boolean	
"kcvZero": false,	boolean	
"sigNone": false,	boolean	
"rsassaPkcs1V15": false,	boolean	
"rsassaPss": false	boolean	
},		
"hashAlgorithm": {	object	Yes
"sha1": false,	boolean	
"sha256": false	boolean	
}		
},		
"S": {	object	
See V properties.		
}		
},		
"R": {	object	
See T properties.		
}		
},		
"S0": {	object	
See M0 properties.		
}		
}		

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

Payload	Type	Required
`},`		
`"keyboard": {`	object	
`"autoBeep": {`	object	Yes
`"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	
`"float": {`	object	
`"x": false,`	boolean	
`"y": false`	boolean	
`}`		
`}]`		
`},`		
`"textTerminal": {`	object	
`"type": "fixed",`	string	Yes
`"resolutions": [{`	array (object)	Yes
`"sizeX": 0,`	integer	
`"sizeY": 0`	integer	
`}],`		
`"keyLock": false,`	boolean	Yes
`"cursor": false,`	boolean	Yes
`"forms": false`	boolean	Yes
`},`		
`"printer": {`	object	
`"type": {`	object	
`"receipt": false,`	boolean	
`"passbook": false,`	boolean	
`"journal": false,`	boolean	
`"document": false,`	boolean	

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

Payload	Type	Required
"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": false,	boolean	
"readEmptyLine": false	boolean	
},		
"writeForm": {	object	
"text": false,	boolean	
"graphics": false,	boolean	
"ocr": false,	boolean	
"micr": false,	boolean	
"msf": false,	boolean	
"barcode": false,	boolean	
"stamp": false	boolean	
},		
"extents": {	object	
"horizontal": false,	boolean	
"vertical": false	boolean	
},		
"control": {	object	
"eject": false,	boolean	
"perforate": false,	boolean	
"cut": false,	boolean	
"skip": false,	boolean	
"flush": false,	boolean	
"retract": false,	boolean	

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

Payload	Type	Required
"stack": false,	boolean	
"partialCut": false,	boolean	
"alarm": false,	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	

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

Payload	Type	Required
`},`		
`"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	
`},`		
`"cardEmbosser": {`	object	
`"compareMagneticStripe": false,`	boolean	
`"magneticStripeRead": false,`	boolean	

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

Payload	Type	Required
"magneticStripeWrite": false,	boolean	
"chipIO": false,	boolean	
"chipProtocol": {	object	
"notSupported": false,	boolean	
"chipT0": false,	boolean	
"chipT1": false,	boolean	
"chipProtocolNotRequired": false	boolean	
},		
"charSupport": ,	undefined	
"ascii": false,	boolean	
"unicode": false	boolean	
"type": {	object	
"emboss": false,	boolean	
"print": false	boolean	
}		
},		
"barcodeReader": {	object	
"canFilterSymbologies": false,	boolean	
"symbologies": {	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	

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

Payload	Type	Required
"itf": false,	boolean	
"code11": false,	boolean	
"code39": false,	boolean	
"code49": false,	boolean	
"code93": false,	boolean	
"code128": false,	boolean	
"msi": false,	boolean	
"plessey": false,	boolean	
"std2Of5": false,	boolean	
"std2Of5Iata": 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	

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

Payload	Type	Required
"japanesePost": false,	boolean	
"chinesePost": false,	boolean	
"koreanPost": false	boolean	
}		
},		
"biometric": {	object	
"type": {	object	Yes
"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	
},		
"maxCapture": 0,	integer	Yes
"templateStorage": Add example to YAML,	string	Yes
"dataFormats": {	object	Yes
"isoFid": false,	boolean	
"isoFmd": false,	boolean	
"ansiFid": false,	boolean	
"ansiFmd": false,	boolean	
"gso": false,	boolean	
"wso": false,	boolean	
"reservedRaw1": false,	boolean	
"reservedTemplate1": false,	boolean	
"reservedRaw2": false,	boolean	
"reservedTemplate2": false,	boolean	
"reservedRaw3": false,	boolean	
"reservedTemplate3": false	boolean	
},		
"encryptionalAlgorithm": {	object	
"ecb": false,	boolean	

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

Payload	Type	Required
"cbc": false,	boolean	
"cfb": false,	boolean	
"rsa": false	boolean	
},		
"storage": {	object	Yes
"secure": false,	boolean	
"clear": false	boolean	
},		
"persistenceModes": {	object	Yes
"persist": false,	boolean	
"clear": false	boolean	
},		
"matchSupported": "none",	string	Yes
"scanModes": {	object	Yes
"scan": false,	boolean	
"match": false	boolean	
},		
"compareModes": {	object	Yes
"verify": false,	boolean	
"identity": false	boolean	
},		
"clearData": {	object	Yes
"scannedData": false,	boolean	
"importedData": false,	boolean	
"setMatchedData": false	boolean	
}		
},		
"camera": {	object	
"type": "cam",	string	
"cameras": {	object	
"room": false,	boolean	
"person": false,	boolean	
"exitSlot": false,	boolean	
"*exampleProperty1*": false,	boolean	
"*exampleProperty2": false*	boolean	
},		
"maxPictures": 0,	integer	

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

Payload	Type	Required
"camData": {	object	
"autoAdd": false,	boolean	
"manAdd": false	boolean	
},		
"maxDataLength": 0,	integer	
"pictureFile": false	boolean	
},		
"dk": {	object	
"hsmVendor": Add example to YAML,	string	
"hsmJournaling": false	boolean	Yes
},		
"lights": {	object	
"cardReader": {	object	
"flashRate": {	object	
"off": false,	boolean	
"slow": ,	undefined	
"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	
},		
"position": {	object	
"left": false,	boolean	
"right": false,	boolean	
"center": false,	boolean	

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

Payload	Type	Required
"top": false,	boolean	
"bottom": false,	boolean	
"front": false,	boolean	
"rear": false	boolean	
}		
},		
"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	

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

Payload	Type	Required
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.		
},		
"statusGood": {	object	
See cardReader properties.		
},		
"statusWarning": {	object	
See cardReader properties.		
},		
"statusBad": {	object	
See cardReader properties.		
},		
"fasciaLight": {	object	
See cardReader properties.		
},		
"*exampleProperty1*": {	object	
See cardReader properties.		
},		
"*exampleProperty2": {*	object	
See exampleProperty1 properties.		
}		
},		
"auxiliaries": {	object	

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

Payload	Type	Required
"operatorSwitch": {	object	
"run": false,	boolean	
"maintenance": false,	boolean	
"supervisor": false	boolean	
},		
"tamperSensor": false,	boolean	
"intTamperSensor": false,	boolean	
"seismicSensor": false,	boolean	
"heatSensor": false,	boolean	
"proximitySensor": false,	boolean	
"ambientLightSensor": false,	boolean	
"enhancedAudioSensor": {	object	
"manual": false,	boolean	
"auto": false,	boolean	
"semiAuto": false,	boolean	
"bidirectional": false	boolean	
},		
"bootSwitchSensor": false,	boolean	
"displaySensor": false,	boolean	
"operatorCallButtonSensor": false,	boolean	
"handsetSensor": {	object	
"manual": false,	boolean	
"auto": false,	boolean	
"semiAuto": false,	boolean	
"microphone": false	boolean	
},		
"headsetMicrophoneSensor": {	object	
"manual": false,	boolean	
"auto": false,	boolean	
"semiAuto": false	boolean	
},		
"fasciaMicrophoneSensor": false,	boolean	
"safeDoor": {	object	
"closed": false,	boolean	
"open": false,	boolean	
"locked": false,	boolean	
"bolted": false,	boolean	

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

Payload	Type	Required
"tampered": false	boolean	
},		
"vandalShield": {	object	
"closed": false,	boolean	
"open": false,	boolean	
"locked": false,	boolean	
"service": false,	boolean	
"keyboard": false,	boolean	
"tampered": false	boolean	
},		
"frontCabinet": {	object	
See safeDoor properties.		
},		
"rearCabinet": {	object	
See safeDoor properties.		
},		
"leftCabinet": {	object	
See safeDoor properties.		
},		
"rightCabinet": {	object	
See safeDoor properties.		
},		
"openCloseIndicator": false,	boolean	
"audio": false,	boolean	
"heating": false,	boolean	
"consumerDisplayBacklight": false,	boolean	
"signageDisplay": false,	boolean	
"volume": 1,	integer	
"ups": {	object	
"low": false,	boolean	
"engaged": false,	boolean	
"powering": false,	boolean	
"recovered": false	boolean	
},		
"audibleAlarm": false,	boolean	
"enhancedAudioControl": {	object	
"headsetDetection": false,	boolean	

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

Payload	Type	Required
`"modeControllable": false`	boolean	
`},`		
`"enhancedMicrophoneControlState": {`	object	
`"headsetDetection": false,`	boolean	
`"modeControllable": false`	boolean	
`},`		
`"microphoneVolume": 1,`	integer	
`"autoStartupMode": {`	object	
`"specific": false,`	boolean	
`"daily": false,`	boolean	
`"weekly": false`	boolean	
`}`		
`},`		
`"vendorApplication": {`	object	
`"supportedAccessLevels": {`	object	
`"basic": false,`	boolean	
`"intermediate": false,`	boolean	
`"full": false,`	boolean	
`"accessNotSupported": false`	boolean	
`}`		
`}`		
`}`		

Properties
interfaces Array of interfaces supported by this XFS4IoT service.

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

Properties
interfaces/name Name of supported XFS4IoT interface. Following values are supported: • `Common` - Common interface. Every device implements this interface. • `CardReader` - CardReader interface. • `CashAcceptor` - CashAcceptor interface. • `CashDispenser` - CashDispenser interface. • `CashManagement` - CashManagement interface. • `PinPad` - PinPad interface. • `Crypto` - Crypto interface. • `KeyManagement` - KeyManagement interface. • `Keyboard` - Keyboard interface. • `TextTerminal` - TextTerminal interface. • `Printer` - Printer interface. • `CardEmbosser` - CardEmbosser interface. • `BarcodeReader` - BarcodeReader interface. • `Lights` - Lights interface. • `Auxiliaries` - Auxiliaries interface. • `VendorMode` - VendorMode interface. • `VendorApplication` - VendorApplication interface.
interfaces/commands The commands supported by the service.
interfaces/commands/CardReader.ReadRawData (example name) A command name. Property name constraints: `pattern: ^[0-9A-Za-z]*\.[0-9A-Za-z]*$`
interfaces/commands/CardReader.ReadRawData/versions The versions of the command supported by the service. There will be one item for each major version supported. The minor version number qualifies the exact version of the message the service supports. Property value constraints: `pattern: ^[1-9][0-9]*\.([1-9][0-9]*|0)$`
interfaces/events The events (both event and unsolicited) supported by the service.
interfaces/events/CardReader.MediaInsertedEvent (example name) An event name. Property name constraints: `pattern: ^[0-9A-Za-z]*\.[0-9A-Za-z]*$`
interfaces/events/CardReader.MediaInsertedEvent/versions The versions of the event supported by the service. There will be one item for each major version supported. The minor version number qualifies the exact version of the message the service supports. Property value constraints: `pattern: ^[1-9][0-9]*\.([1-9][0-9]*|0)$`
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.

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

Properties
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 on 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/softwareName Specifies the software name. The property is omitted, if the software name is unknown.
common/deviceInformation/software/softwareVersion Specifies the software version. The property is omitted, if the software version 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.

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

Properties
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 Application. If omitted, no Execute commands can be accepted.
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.
common/commandNonceTimeout If this device supports end to end security and can return a command nonce with the command Common.GetCommandNonce, and the hardware automatically clears the command nonce after a fixed length of time, this property will report the number of seconds between returning the command nonce and clearing it. The value is given in seconds but it should not be assumed that the timeout will be accurate to the nearest second. The nonce may also become invalid before the timeout, for example because of a power failure. Hardware may impose a timeout to reduce the chance of an attacker re-using a nonce value or a token. This timeout will be long enough to support normal operations such as dispense and present including creating the required token on the host and passing it to the hardware. For example, a command nonce might time out after one hour (that is, 3600 seconds). If commandNonceTimeout is not reported, or it has a value of zero, then the command nonce will never timeout. It may still become invalid, for example because of a power failure or when explicitly cleared.
cardReader Capability information for XFS4IoT services implementing the CardReader interface. This will be omitted if the CardReader interface is not supported.

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

Properties
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.Move 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.

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

Properties
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/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): • `notSupported` - The device does not support power on or off options. • `exit` - The card will be moved to the exit position. • `retain` - The card will moved to a *retain* storage unit. • `exitThenRetain` - The card will be moved to the exit position for a finite time, then if not taken, the card will be moved to a *retain* storage unit. The time for which the card remains at the exit position is vendor dependent. • `transport` - The card will be moved to the transport position. If multiple *retain* storage units are present, the storage unit to which the card is retained is vendor specific.

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

Properties
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/readWriteAccessFromExit Specifies whether a card may be read or written after having been moved to to the exit position with a CardReader.Move command. The card will be moved 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 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/positions Specifies the target positions that is supported for the CardReader.MoveCard command. This is independent of the storage units.
cardReader/positions/exit The device can move a card to the exit position. In this position, the card is accessible to the user.

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

Properties
cardReader/positions/transport The device can move a card to the transport. In this position, the card is not accessible to the user. A service which supports this position must also support the *exit* position.
cardReader/cardTakenSensor Specifies whether or not the card reader has the ability to detect when a card is taken from the exit slot by a user. If true, a CardReader.MediaTakenEvent will be sent when the card is removed.
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.CashInStart command. Property value constraints: `minimum: 1`
cashAcceptor/shutter If 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 application control of the shutter.
cashAcceptor/shutterControl If true the shutter is controlled implicitly by the Service. If false the shutter must be controlled explicitly by the application using the OpenShutter and the CloseShutter commands. In either case the PresentMedia command may be used if the presentControl field is false. The shutterControl field is always 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 true the Service generates an accompanying CashAcceptor.ItemsTaken event. If 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 true the Service generates an accompanying CashAcceptor.ItemsInserted event. If 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.

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

Properties
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.

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

Properties
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 the CashAcceptor.CompareSignature command.
cashAcceptor/replenish If set to true the CashAcceptor.ReplenishTarget and CashAcceptor.Replenish commands are supported.

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

Properties
cashAcceptor/cashInLimit Specifies whether the cash-in limitation is supported or not for the CashAcceptor.CashInStart 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/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 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 false, the CashAcceptor will not support the physical lock/unlock control of the CashAcceptor device or the cash units.
cashAcceptor/deplete If set to true the CashAcceptor.Deplete command is supported.
cashAcceptor/counterfeitAction Specifies whether level 2/3 notes (see NoteClassification) are allowed to be returned to the customer during a cash-in transaction. If 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.

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

Properties
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 application 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.

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

Properties
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.

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

Properties
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 application 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 application 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 storage unit exchange operations supported by the device.
cashManagement/exchangeType/byHand The device supports manual replenishment either by filling the cash storage unit by hand or by replacing the cash storage unit.
cashManagement/exchangeType/toCassettes The device supports moving items from the replenishment cash storage unit to another cash storage unit.
cashManagement/exchangeType/clearRecycler The device supports the emptying of recycle cash storage units.
cashManagement/exchangeType/depositInto The device supports moving items from the deposit entrance to the bill cash storage 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.

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

Properties
cashManagement/physicalNoteList Specifies whether the Service supports note number lists on physical storage 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/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.

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

Properties
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 doesnâ€™t apply for self-service).
pinPad/idcConnect 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 Key-value pair of attributes supported by the PinPad.GetPinblock command to generate encrypted PIN block.

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

Properties
pinPad/pinBlockAttributes/P0 (example name) Specifies the key usages supported by the PinPad.PinBlock command. The following values are possible: • `P0` - PIN Encryption Property name constraints: `pattern: ^P0$`
pinPad/pinBlockAttributes/P0/T (example name) Specifies the encryption algorithms supported by the PinPad.PinBlock command. The following values are possible: • `A` - AES. • `D` - DEA. • `R` - RSA. • `T` - Triple DEA (also referred to as TDEA). Property name constraints: `pattern: ^[ADT]$`
pinPad/pinBlockAttributes/P0/T/E (example name) Specifies the encryption modes supported by the PinPad.PinBlock command. The following value is possible: • `E` - Encrypt Property name constraints: `pattern: ^E$`
pinPad/pinBlockAttributes/P0/T/E/cryptoMethod Specifies the cryptographic method supported. If the algorithm is 'A', 'D', or 'T', then the following properties can be true." • `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 the algorithm is 'R', then following properties can be true. • `rsaesPkcs1V15` - Use the RSAES_PKCS1-v1.5 algorithm. • `rsaesOaep` - Use the RSAES OAEP algorithm.
pinPad/pinBlockAttributes/P0/T/E/cryptoMethod/ecb The ECB encryption method.
pinPad/pinBlockAttributes/P0/T/E/cryptoMethod/cbc The CBC encryption method.
pinPad/pinBlockAttributes/P0/T/E/cryptoMethod/cfb The CFB encryption method.
pinPad/pinBlockAttributes/P0/T/E/cryptoMethod/ofb The The OFB encryption method.
pinPad/pinBlockAttributes/P0/T/E/cryptoMethod/ctr The CTR method defined in NIST SP800-38A.
pinPad/pinBlockAttributes/P0/T/E/cryptoMethod/xts The XTS method defined in NIST SP800-38E.

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

Properties
pinPad/pinBlockAttributes/P0/T/E/cryptoMethod/rsaesPkcs1V15 The RSAES_PKCS1-v1.5 algorithm.
pinPad/pinBlockAttributes/P0/T/E/cryptoMethod/rsaesOaep The RSAES OAEP algorithm.
crypto Capability information for XFS4IoT services implementing the Crypto interface. This will be omitted if the Crypto interface is not supported.
crypto/emvHashAlgorithm Specifies which hash algorithm is supported for the calculation of the HASH.
crypto/emvHashAlgorithm/sha1Digest The SHA 1 digest algorithm is supported by the Crypto.Digest command.
crypto/emvHashAlgorithm/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 Key-value pair of attributes supported by the Crypto.CryptoData command to encrypt or decrypt data.
crypto/cryptoAttributes/D0 (example name) The following key usage is possible: • `D0` - Symmetric data encryption. • `D1` - Asymmetric data encryption. Property name constraints: `pattern: ^D[0-1]$`
crypto/cryptoAttributes/D0/D (example name) Specifies the encryption algorithms supported by Crypto.CryptoData command. The following values are possible: • `A` - AES. • `D` - DEA. • `R` - RSA. • `T` - Triple DEA (also referred to as TDEA). Property name constraints: `pattern: ^[ADRT]$`
crypto/cryptoAttributes/D0/D/D (example name) Specifies the encryption mode supported by Crypto.CryptoData command. The following values are possible: • `D` - Decrypt • `E` - Encrypt Property name constraints: `pattern: ^[DE]$`

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

Properties
crypto/cryptoAttributes/D0/D/D/cryptoMethod Specifies the cryptographic method supported by the Crypto.CryptoData command. If the key usage is any of the MAC usages (i.e. 'M1'), then the following properties can be true. • `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 the algorithm is 'R' and the key usage is 'D0', then the following properties can be true. • `rsaesPkcs1V15` - RSAES_PKCS1-v1.5 algorithm. • `rsaesOaep` - The RSAES OAEP algorithm.
crypto/cryptoAttributes/D0/D/D/cryptoMethod/ecb The ECB encryption method.
crypto/cryptoAttributes/D0/D/D/cryptoMethod/cbc The CBC encryption method.
crypto/cryptoAttributes/D0/D/D/cryptoMethod/cfb The CFB encryption method.
crypto/cryptoAttributes/D0/D/D/cryptoMethod/ofb The The OFB encryption method.
crypto/cryptoAttributes/D0/D/D/cryptoMethod/ctr The CTR method defined in NIST SP800-38A.
crypto/cryptoAttributes/D0/D/D/cryptoMethod/xts The XTS method defined in NIST SP800-38E.
crypto/cryptoAttributes/D0/D/D/cryptoMethod/rsaesPkcs1V15 The RSAES_PKCS1-v1.5 algorithm.
crypto/cryptoAttributes/D0/D/D/cryptoMethod/rsaesOaep The RSAES OAEP algorithm.
crypto/authenticationAttributes Key-value pair of attributes supported by the Crypto.GenerateAuthentication command to generate authentication data.

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

Properties
crypto/authenticationAttributes/S0 (example name) The following key usages are possible: • `S0` - Asymmetric key pair for digital signature. • `S1` - Asymmetric key pair, CA. • `S2` - Asymmetric key pair, nonX9.24 key. • `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` - 9797-1:2011 MAC Algorithm 5/CMAC. • `M7` - HMAC. • `M8` - ISO 9797-1:2011 MAC Algorithm 6. Property name constraints: `pattern: ^M[0-8]$\|^S[0-2]$`
crypto/authenticationAttributes/S0/R (example name) Specifies the encryption algorithms supported by the Crypto.GenerateAuthentication command. The following value is possible: • `A` - AES. • `D` - DEA. • `R` - RSA. • `T` - Triple DEA (also referred to as TDEA). Property name constraints: `pattern: ^[ADRT]$`
crypto/authenticationAttributes/S0/R/G (example name) Specifies the encryption mode supported by Crypto.GenerateAuthentication command. The following values are possible: • `C` - Both Generate and Verify. • `G` - Generate. This be used to generate a MAC. • `S` - Signature • `T` - Both Sign and Decrypt. Property name constraints: `pattern: ^[CGST]$`
crypto/authenticationAttributes/S0/R/G/cryptoMethod Specifies the asymmetric signature verification method supported by the Crypto.GenerateAuthentication command. If the key usage is any of the MAC usages (i.e. 'M1'), then following properties are false.
crypto/authenticationAttributes/S0/R/G/cryptoMethod/rsassaPkcs1V15 The RSASSA-PKCS1-v1.5 algorithm.
crypto/authenticationAttributes/S0/R/G/cryptoMethod/rsassaPss The the RSASSA-PSS algorithm.
crypto/authenticationAttributes/S0/R/G/hashAlgorithm Specifies the hash algorithm supported.

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

Properties
crypto/authenticationAttributes/S0/R/G/hashAlgorithm/sha1 The SHA 1 digest algorithm. If the key usage is any of the MAC usages (i.e. 'M1'), then following properties are false.
crypto/authenticationAttributes/S0/R/G/hashAlgorithm/sha256 The SHA 256 digest algorithm, as defined in ISO/IEC 10118-3:2004 and FIPS 180-2.
crypto/verifyAttributes Key-value pair of attributes supported by the Crypto.VerifyAuthentication command to verify authentication data.
crypto/verifyAttributes/S0 (example name) The following key usages are possible: • `S0` - Asymmetric key pair for digital signature. • `S1` - Asymmetric key pair, CA. • `S2` - Asymmetric key pair, nonX9.24 key. • `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` - 9797-1:2011 MAC Algorithm 5/CMAC. • `M7` - HMAC. • `M8` - ISO 9797-1:2011 MAC Algorithm 6. Property name constraints: `pattern: ^M[0-8]$\|^S[0-2]$`
crypto/verifyAttributes/S0/R (example name) Specifies the encryption algorithms supported by Crypto.VerifyAuthentication command. The following value is possible: • `A` - AES. • `D` - DEA. • `R` - RSA. • `T` - Triple DEA (also referred to as TDEA). Property name constraints: `pattern: ^ADRT$`
crypto/verifyAttributes/S0/R/V (example name) Specifies the encryption mode supported by Crypto.VerifyAuthentication command. The following values are possible: • `V` - Verify. This be used to verify a MAC. Property name constraints: `pattern: ^V$`
crypto/verifyAttributes/S0/R/V/cryptoMethod Specifies the asymmetric signature verification method supported by the Crypto.VerifyAuthentication command. If the key usage is any of the MAC usages (i.e. 'M1'), then following properties are false.
crypto/verifyAttributes/S0/R/V/cryptoMethod/rsassaPkcs1V15 The RSASSA-PKCS1-v1.5 algorithm.

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

Properties
crypto/verifyAttributes/S0/R/V/cryptoMethod/rsassaPss The the RSASSA-PSS algorithm.
crypto/verifyAttributes/S0/R/V/hashAlgorithm Specifies the hash algorithm supported. If the key usage is any of the MAC usages (i.e. 'M1'), then following properties are false.
crypto/verifyAttributes/S0/R/V/hashAlgorithm/sha1 The SHA 1 digest algorithm.
crypto/verifyAttributes/S0/R/V/hashAlgorithm/sha256 The SHA 256 digest algorithm, as defined in ISO/IEC 10118-3:2004 and FIPS 180-2.
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/derivationAlgorithms Supported derivation algorithms.
keyManagement/derivationAlgorithms/chipZka Algorithm for the derivation of a chip card individual key as described by the German ZKA.
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.

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

Properties
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/emvImportSchemes Identifies the supported emv Import Scheme(s).
keyManagement/emvImportSchemes/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.

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

Properties
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.

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

Properties
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/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 Key-value pair of attributes supported by KeyManagement.ImportKey command for the key to be loaded.

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

Properties

keyManagement/keyAttributes/M0 (example name)

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` - EMV / Chip Issuer Master Key: Application 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 5.
- `M6` - ISO 9797-1:2011 MAC Algorithm 5 / CMAC.
- `M7` - HMAC.
- `M8` - ISO 9797-1:2011 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.
- `00 – 99` - These numeric values are reserved for proprietary use.

Property name constraints:

```
pattern: ^B[0-2]$|^C0$|^D[0-2]$|^E[0-6]$|^I0$|^K[0-3]$|^M[0-8]$|^P0$|^S[0-2]$|^V[0-4]$|^[0-9][0-9]$
```

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

Properties
keyManagement/keyAttributes/M0/T (example name) Specifies the encryption algorithms supported by the KeyManagement.ImportKey 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. Property name constraints: `pattern: ^[0-9ADRT]$`
keyManagement/keyAttributes/M0/T/C (example name) Specifies the encryption modes supported by KeyManagement.ImportKey key. The following values are possible: • `B` - Both Encrypt and Decrypt / Wrap and unwrap. • `C` - Both Generate and Verify. • `D` - Decrypt / Unwrap Only. • `E` - Encrypt / Wrap Only. • `G` - Generate Only. • `S` - Signature Only. • `T` - Both Sign and Decrypt. • `V` - Verify Only. • `X` - Key used to derive other keys(s). • `Y` - Key used to create key variants. • `0 – 9` - These numeric values are reserved for proprietary use. Property name constraints: `pattern: ^[0-9BCDEGSTVXY]$`

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

Properties

keyManagement/keyAttributes/M0/T/C/restricted

Specifies restricted key usage of the key associated with the key usage. This property can be omitted if there is no restricted key usage required. Following restricted key usage can be set if the key Usage is either 'K0' or 'K1'. 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` - EMV / Chip Issuer Master Key: Application 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).
- `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 5.
- `M6` - ISO 9797-1:2011 MAC Algorithm 5 / CMAC.
- `M7` - HMAC.
- `M8` - ISO 9797-1:2011 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.
- `00 – 99` - These numeric values are reserved for proprietary use.

Property value constraints:

```
pattern: ^B[0-2]$|^C0$|^D[0-2]$|^E[0-6]$|^I0$|^K[2-3]$|^M[0-8]$|^P0$|^S[0-2]$|^V[0-4]$|^[0-9][0-9]$
```

keyManagement/decryptAttributes

Key-value pair of attributes supported by the KeyManagement.ImportKey command for the key used to decrypt or unwrap the key being imported.

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

Properties
keyManagement/decryptAttributes/A (example name) Specifies the encryption algorithms supported by the KeyManagement.ImportKey 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. Property name constraints: `pattern: ^[0-9ADRT]$`
keyManagement/decryptAttributes/A/decryptMethod Specifies the cryptographic method supported. If the algorithm is 'A', 'D', or 'T', then one of following property must be true and both rsaesPkcs1V15, rsaesOaep properties are false. • `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 the algorithm is 'R', then one of following property must be true and ecb, cbc, cfb, ofb, ctr, xts must be all false. • `rsaesPkcs1V15` - Use the RSAES_PKCS1-v1.5 algorithm. • `rsaesOaep` - Use the RSAES OAEP algorithm.
keyManagement/decryptAttributes/A/decryptMethod/ecb The ECB encryption method.
keyManagement/decryptAttributes/A/decryptMethod/cbc The CBC encryption method.
keyManagement/decryptAttributes/A/decryptMethod/cfb The CFB encryption method.
keyManagement/decryptAttributes/A/decryptMethod/ofb The The OFB encryption method.
keyManagement/decryptAttributes/A/decryptMethod/ctr The CTR method defined in NIST SP800-38A.
keyManagement/decryptAttributes/A/decryptMethod/xts The XTS method defined in NIST SP800-38E.
keyManagement/decryptAttributes/A/decryptMethod/rsaesPkcs1V15 The RSAES-PKCS1-v1.5 algorithm.
keyManagement/decryptAttributes/A/decryptMethod/rsaesOaep The RSAES-OAEP algorithm.
keyManagement/verifyAttributes Key-value pair of attributes supported by the KeyManagement.ImportKey for the key used for verification before importing the key.

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

Properties
keyManagement/verifyAttributes/M0 (example name) Specifies the key usages supported by the KeyManagement.ImportKey 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` - ISO 9797-1:2011 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. Property name constraints: `pattern: ^M[0-8]$\|^S[0-2]$|^[0-9][0-9]$`
keyManagement/verifyAttributes/M0/T (example name) Specifies the encryption algorithms supported by the KeyManagement.ImportKey command. The following values are possible: • `A` - AES. • `D` - DEA. • `R` - RSA. • `T` - Triple DEA (also referred to as TDEA). • `0 – 9` - These numeric values are reserved for proprietary use. Property name constraints: `pattern: ^[0-9ADRT]$`
keyManagement/verifyAttributes/M0/T/V (example name) Specifies the encryption modes supported by the KeyManagement.ImportKey command. The following values are possible: • `S` - Signature. • `V` - Verify Only. • `0 – 9` - These numeric values are reserved for proprietary use. Property name constraints: `pattern: ^[0-9SV]$`

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

Properties
keyManagement/verifyAttributes/M0/T/V/cryptoMethod This parameter specifies the cryptographic method that will be used with encryption algorithm. If the algorithm is 'A', 'D', or 'T' and the key usage is a MAC usage (i.e. 'M1'), then all properties are false. If the algorithm is 'A', 'D', or 'T' and the key usage is '00', then one of properties must be set true. • `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 the algorithm is 'R' and the key usage is not '00', then one of properties must be set true. • `sigNone` - No signature algorithm specified. No signature verification will take place and the content of verificationData must be set. • `rsassaPkcs1V15` - Use the RSASSA-PKCS1-v1.5 algorithm. • `rsassaPss` - Use the RSASSA-PSS algorithm.
keyManagement/verifyAttributes/M0/T/V/cryptoMethod/kcvNone The ECB encryption method.
keyManagement/verifyAttributes/M0/T/V/cryptoMethod/kcvSelf There is no key check value verification required.
keyManagement/verifyAttributes/M0/T/V/cryptoMethod/kcvZero The key check value (KCV) is created by encrypting a zero value with the key.
keyManagement/verifyAttributes/M0/T/V/cryptoMethod/sigNone The No signature algorithm specified. No signature verification will take place.
keyManagement/verifyAttributes/M0/T/V/cryptoMethod/rsassaPkcs1V15 The RSASSA-PKCS1-v1.5 algorithm.
keyManagement/verifyAttributes/M0/T/V/cryptoMethod/rsassaPss The RSASSA-PSS algorithm.
keyManagement/verifyAttributes/M0/T/V/hashAlgorithm For asymmetric signature verification methods (key usage is 'S0', 'S1', or 'S2'), then one of the following properties are true. If the key usage is any of the MAC usages (i.e. 'M1'), then properties both 'sha1' and 'sha256' are false.
keyManagement/verifyAttributes/M0/T/V/hashAlgorithm/sha1 The SHA 1 digest algorithm.
keyManagement/verifyAttributes/M0/T/V/hashAlgorithm/sha256 The SHA 256 digest algorithm, as defined in ISO/IEC 10118-3:2004 and FIPS 180-2.
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.

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

Properties
keyboard/autoBeep/activeSelectable Automatic beeping for active keys can be controlled turned on and off by the application. If this flag is not set then automatic beeping for active keys cannot be controlled by an application.
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 application. If this flag is not set then automatic beeping for in-active keys cannot be controlled by an application.
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/float Specifies if the device can float the touch keyboards. Both properties *x* and *y* are false if the device cannot randomly shift the layout.
keyboard/etsCaps/float/x Specifies that the device will randomly shift the layout in a horizontal direction
keyboard/etsCaps/float/y Specifies that the 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 as one of the following flags: • `fixed` - The text terminal unit is a fixed device. • `removable` - The text terminal unit is a removable device.

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

Properties
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). Property value constraints: `minimum: 0`
textTerminal/resolutions/sizeY Specifies the vertical size of the display of the text terminal unit (the number of rows that can be displayed). Property value constraints: `minimum: 0`
textTerminal/keyLock Specifies whether the text terminal unit has a key lock switch.
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.
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 application 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.

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

Properties
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.

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

Properties
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 application 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.

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

Properties
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.

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

Properties
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. Applications 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.

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

Properties
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.
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.

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

Properties
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 application 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

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

Properties
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

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

Properties
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

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

Properties
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.

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

Properties
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/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 [Ref. biometric-4].
biometric/dataFormats/isoFmd ISO FMD template format [Ref. biometric-5].
biometric/dataFormats/ansiFid Raw ANSI FID format [Ref. biometric-2].
biometric/dataFormats/ansiFmd ANSI FMD template format [Ref. biometric-3].
biometric/dataFormats/qso Raw QSO image format

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

Properties
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/encryptionalAlgorithm Supported encryption algorithms or cryptNone if no encryption algorithms
biometric/encryptionalAlgorithm/ecb Triple DES with Electronic Code Book.
biometric/encryptionalAlgorithm/cbc Triple DES with Cipher Block Chaining
biometric/encryptionalAlgorithm/cfb Triple DES with Cipher Feed Back.
biometric/encryptionalAlgorithm/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.

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

Properties
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.
camera Capability information for XFS4IoT services implementing the Camera interface. This will be omitted if the Camera interface is not supported.
camera/type Specifies the type of the camera device; only current value is • `cam` - Camera system.
camera/cameras Specifies whether cameras are available. The value of this parameter is either TRUE or FALSE. TRUE is available.

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

Properties
camera/cameras/room Specifies whether the camera that monitors the whole self-service area is available.
camera/cameras/person Specifies whether the camera that monitors the person standing in front of the self-service is available.
camera/cameras/exitSlot Specifies whether the camera that monitors the exit slot(s) of the self-service machine is available.
camera/maxPictures Specifies the maximum number of pictures that can be stored on the recording media. Property value constraints: `minimum: 0`
camera/camData Specifies whether the methods are supported for adding data to the picture. Ture means the method is supported. False indicates that the method is not supported. If all methods are false, No data can be added to the picture.
camera/camData/autoAdd Specifies whether data can be added automatically to the picture.
camera/camData/manAdd Specifies whether data can be added manually to the picture using the parameter Camera.TakePicture.camData.
camera/maxDataLength Specifies the maximum length of the data that is displayed on the photo. Zero, if data cannot be manually added to the picture. Property value constraints: `minimum: 0`
camera/pictureFile Specifies whether the parameter [Camera.TakePicture.pictureFile](#camera.takepicture.command.properties.picturefile) is supported. Ture means the parameter is supported. False indicates that the parameter is not supported.
dk Capability information for XFS4IoT services implementing the DK interface. This will be omitted if the DK interface is not supported.
dk/hsmVendor Identifies the hsm Vendor. hsmVendor is an empty string or this property is omitted when the hsm Vendor is unknown or the HSM is not supported.
dk/hsmJournaling Specifies whether the hsm supports journaling by the DK.GetJournal command. The value of this parameter is either TRUE or FALSE. TRUE means the hsm supports journaling by DK.GetJournal.
lights Capability information for XFS4IoT services implementing the Lights interface. This will be omitted if the Lights interface is not supported.
lights/cardReader Card Unit Light.
lights/cardReader/flashRate Indicates the light flash rate.

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

Properties
lights/cardReader/flashRate/off The light can be turned off.
lights/cardReader/flashRate/slow The light can flash slowly.
lights/cardReader/flashRate/medium The light can flash medium frequency.
lights/cardReader/flashRate/quick The light can flash quickly.
lights/cardReader/flashRate/continuous The light can flash continuous (steady).
lights/cardReader/color Indicates the light color.
lights/cardReader/color/red The light can be red.
lights/cardReader/color/green The light can be green.
lights/cardReader/color/yellow The light can be yellow.
lights/cardReader/color/blue The light can be blue.
lights/cardReader/color/cyan The light can be cyan.
lights/cardReader/color/magenta The light can be magenta.
lights/cardReader/color/white The light can be white .
lights/cardReader/direction Indicates the light direction.
lights/cardReader/direction/entry The light can indicate entry.
lights/cardReader/direction/exit The light can indicate exit.
lights/cardReader/position Indicates the light position.
lights/cardReader/position/left The left position.
lights/cardReader/position/right The right position.
lights/cardReader/position/center The center position.

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

Properties
lights/cardReader/position/top The top position.
lights/cardReader/position/bottom The bottom position.
lights/cardReader/position/front The front position.
lights/cardReader/position/rear The rear position.
lights/pinPad Pin Pad Light.
lights/notesDispenser Notes Dispenser Light.
lights/coinDispenser Coin Dispenser Light.
lights/receiptPrinter Receipt Printer Light.
lights/passbookPrinter Passbook Printer Light.
lights/EnvelopeDepository Envelope Depository Light.
lights/chequeUnit Cheque Unit Light.
lights/billAcceptor Bill Acceptor Light.
lights/envelopeDispenser Envelope Dispenser Light.
lights/documentPrinter Document Printer Light.
lights/coinAcceptor Coin Acceptor Light.
lights/scanner Scanner Light.
lights/contactless Contactless Light.
lights/cardUnit2 Card Unit 2 Light.
lights/notesDispenser2 Notes Dispenser 2 Light.
lights/billAcceptor2 Bill Acceptor 2 Light.

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

Properties
lights/statusGood Status indicator light - Good.
lights/statusWarning Status indicator light - Warning.
lights/statusBad Status indicator light - Bad.
lights/fasciaLight Fascia light.
lights/exampleProperty1 (example name)
auxiliaries Capability information for XFS4IoT services implementing the Auxiliaries interface. This will be omitted if the Auxiliaries interface is not supported.
auxiliaries/operatorSwitch Specifies which states the Operator Switch can be set to, if available.
auxiliaries/operatorSwitch/run The switch can be set in Run mode. default: false
auxiliaries/operatorSwitch/maintenance The switch can be set in Maintenance mode. default: false
auxiliaries/operatorSwitch/supervisor The switch can be set in Supervisor mode. default: false
auxiliaries/tamperSensor Specifies whether the Tamper Sensor for the terminal is available. default: false
auxiliaries/intTamperSensor Specifies whether the Internal Tamper Sensor for the terminal is available. default: false
auxiliaries/seismicSensor Specifies whether the Seismic Sensor for the terminal is available. default: false
auxiliaries/heatSensor Specifies whether the Heat Sensor for the terminal is available. default: false
auxiliaries/proximitySensor Specifies whether the Proximity Sensor for the terminal is available. default: false
auxiliaries/ambientLightSensor Specifies whether the Ambient Light Sensor for the terminal is available. default: false

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

Properties
auxiliaries/enhancedAudioSensor Specifies which mode the Audio Jack supports, if present.
auxiliaries/enhancedAudioSensor/manual The Audio Jack is available and supports manual mode. default: false
auxiliaries/enhancedAudioSensor/auto The Audio Jack is available and supports auto mode. default: false
auxiliaries/enhancedAudioSensor/semiAuto The Audio Jack is available and supports semi-auto mode. default: false
auxiliaries/enhancedAudioSensor/bidirectional The Audio Jack is available and can support headphones that have an integrated microphone via a single jack. default: false
auxiliaries/bootSwitchSensor Specifies whether the Boot Switch Sensor for the terminal is available. default: false
auxiliaries/displaySensor Specifies whether the Consumer Display Sensor is available. default: false
auxiliaries/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 application. default: false
auxiliaries/handsetSensor Specifies which mode the Handset supports, if present.
auxiliaries/handsetSensor/manual The Handset is available and it supports manual mode. default: false
auxiliaries/handsetSensor/auto The Handset is available and it supports auto mode. default: false
auxiliaries/handsetSensor/semiAuto The Handset is available and it supports semi-auto mode. default: false
auxiliaries/handsetSensor/microphone The Handset is available and contains an embedded microphone for audio input. default: false
auxiliaries/headsetMicrophoneSensor Specifies whether the Microphone Jack is present, and if so, which modes it supports. If the enhancedAudio capability indicates the presence of a bi-directional Audio Jack then both sensors reference the same physical jack.

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

Properties
auxiliaries/headsetMicrophoneSensor/manual The Microphone Jack is available and it supports manual mode. default: false
auxiliaries/headsetMicrophoneSensor/auto The Microphone Jack is available and it supports auto mode. default: false
auxiliaries/headsetMicrophoneSensor/semiAuto The Microphone Jack is available and it supports semi-auto mode. default: false
auxiliaries/fasciaMicrophoneSensor Specifies whether a Fascia Microphone (for public audio input) is present. default: false
auxiliaries/safeDoor Specifies whether the Safe Door is available, and if so, which states it supports.
auxiliaries/safeDoor/closed Specifies that the door can report the closed state. default: false
auxiliaries/safeDoor/open Specifies that the door can report the open state. default: false
auxiliaries/safeDoor/locked Specifies that the door can report the locked state. default: false
auxiliaries/safeDoor/bolted Specifies that the door can report the bolted state. default: false
auxiliaries/safeDoor/tampered Specifies that the door can report the tampered state. default: false
auxiliaries/vandalShield Specifies the states the Vandal Shield can support, if available.
auxiliaries/vandalShield/closed The Vandal Shield can be closed. default: false
auxiliaries/vandalShield/open The Vandal Shield can be open. default: false
auxiliaries/vandalShield/locked The Vandal Shield can be locked. default: false

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

Properties
auxiliaries/vandalShield/service The Vandal Shield can be in the service position. default: false
auxiliaries/vandalShield/keyboard The Vandal Shield can be in a position that permits access to the keyboard. default: false
auxiliaries/vandalShield/tampered The Vandal Shield can detect potential tampering. default: false
auxiliaries/frontCabinet Specifies whether at least one Front Cabinet Door is available, and if so, which states they support.
auxiliaries/rearCabinet Specifies whether at least one Rear Cabinet Door is available, and if so, which states they support.
auxiliaries/leftCabinet Specifies whether at least one left Cabinet Door is available, and if so, which states they support.
auxiliaries/rightCabinet Specifies whether at least one right Cabinet Door is available, and if so, which states they support.
auxiliaries/openCloseIndicator Specifies whether the Open/Closed Indicator is available. default: false
auxiliaries/audio Specifies whether the Audio Indicator device is available. default: false
auxiliaries/heating Specifies whether the Internal Heating device is available. default: false
auxiliaries/consumerDisplayBacklight Specifies whether the Consumer Display Backlight is available. default: false
auxiliaries/signageDisplay Specifies whether the Signage Display is available. default: false

auxiliaries/volume
Specifies the Volume Control increment/decrement value recommended by the vendor.
Property value constraints:
`minimum: 1`
`maximum: 1000`

auxiliaries/ups Specifies what states the UPS device supports as a combination of the following values:
auxiliaries/ups/low The UPS can indicate that its charge level is low. default: false

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

Properties
auxiliaries/ups/engaged The UPS can be engaged and disengaged by the application. default: false
auxiliaries/ups/powering The UPS can indicate that it is powering the system while the main power supply is off. default: false
auxiliaries/ups/recovered The UPS can indicate that it was engaged when the main power went off. default: false
auxiliaries/audibleAlarm Specifies whether the Audible Alarm is available. default: false
auxiliaries/enhancedAudioControl Specifies the modes the Enhanced Audio Controller can support. The Enhanced Audio Controller controls how private and public audio are broadcast when the headset is inserted into/removed from the audio jack and when the handset is off-hook/on-hook. In the following Privacy Device is used to refer to either the headset or handset. The modes it supports are specified as a combination of the following values:
auxiliaries/enhancedAudioControl/headsetDetection The Enhanced Audio Controller is supports Privacy Device activation/deactivation. The device is able to report events to indicate Privacy Device activation/deactivation. default: false
auxiliaries/enhancedAudioControl/modeControllable The Enhanced Audio Controller supports application control of the Privacy Device mode via the setAuxiliaries command. default: false
auxiliaries/enhancedMicrophoneControlState Specifies the modes the Enhanced Microphone Controller can support. The Enhanced Microphone Controller controls how private and public audio input are transmitted when the headset is inserted into/removed from the audio jack and when the handset is off-hook/on-hook. In the following Privacy Device is used to refer to either the headset or handset. The modes it supports are specified as a combination of the following values:
auxiliaries/enhancedMicrophoneControlState/headsetDetection The Enhanced Microphone Controller supports Privacy Device activation/deactivation. The device is able to report events to indicate Privacy Device activation/deactivation. default: false
auxiliaries/enhancedMicrophoneControlState/modeControllable The Enhanced Microphone Controller supports application control of the Privacy Device mode via the setAuxiliaries command. default: false

auxiliaries/microphoneVolume
Specifies the Microphone Volume Control increment/decrement value recommended by the vendor.
Property value constraints:
`minimum: 1`
`maximum: 1000`

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

Properties
auxiliaries/autoStartupMode Specifies which mode of the auto start-up control is supported. Specified as a combination of the following values:
auxiliaries/autoStartupMode/specific The device supports one-time auto start-up on a specific date at a specific time. default: false
auxiliaries/autoStartupMode/daily TThe device supports auto start-up every day at a specific time. default: false
auxiliaries/autoStartupMode/weekly The device supports auto start-up at a specified time on a specific day of every week. default: false
vendorApplication Capability information for XFS4IoT services implementing the VendorApplication interface. This will be omitted if the Vendor Application interface is not supported.
vendorApplication/supportedAccessLevels Specifies the supported access levels. This allows the application to show a user interface with reduced or extended functionality depending on the access levels. The exact meaning or functionalities definition is left to the vendor. If no access levels are supported this field will be accessNotSupported. Otherwise this field will be set to a combination of true or false value for supported or not supported respectively.
vendorApplication/supportedAccessLevels/basic The application supports the basic access level. Once the application is active it will show the user interface for the basic access level.
vendorApplication/supportedAccessLevels/intermediate The application supports the intermediate access level. Once the application is active it will show the user interface for the intermediate access level.
vendorApplication/supportedAccessLevels/full The application supports the full access level. Once the application is active it will show the user interface for the full access level.
vendorApplication/supportedAccessLevels/accessNotSupported Access levels are not supported.

Event Messages

None

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

3.2.3 Common.SetVersions

This command sets the major versions of the command, event and unsolicited message types for the client connection. The completion message version will match the command message version.

Versions are set only for the client connection on which the command is received. It does not modify the versions other client connections expect to use.

This command can be used while in Vendor Dependent Mode.

Command Message

Payload	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"commands": {`	object	
`"CardReader.ReadRawData": 1,`	integer	
`"CardReader.Move": 1`	integer	
`},`		
`"events": {`	object	
`"CardReader.MediaInsertedEvent": 1,`	integer	
`"CardReader.MediaTakenEvent": 1`	integer	
`}`		
`}`		

Properties

timeout
Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be canceled.
default: 0

commands
The commands for which a version is being set.
Property value constraints:
`minProperties: 1`

commands/CardReader.ReadRawData (example name)
The major version of the command that the Service should use.
Property name constraints:
`pattern: ^[0-9A-Za-z]*\.[0-9A-Za-z]*$`
Property value constraints:
`minimum: 1`

events
The events for which a version is being set
Property value constraints:
`minProperties: 1`

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

Properties

events/CardReader.MediaInsertedEvent (example name)
The major version of the event that the Service should use.
Property name constraints:
`pattern: ^[0-9A-Za-z]*\.[0-9A-Za-z]*$`
Property value constraints:
`minimum: 1`

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	string	
`}`		

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

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

3.2.4 Common.Cancel

This command instructs the service to attempt to cancel one, more or all command requests associated with the client connection on which this command is received.

This command can be used while in Vendor Dependent Mode.

Command Message

Payload	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"requestIds": [1, 2]`	array (integer)	
`}`		

Properties

timeout

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

default: 0

requestIds

The request(s) to be canceled.

If included, the service will only attempt to cancel the specified command requests which are queued or executing and which are associated with the client connection on which this command is received. All other request IDs will be ignored.

If omitted, the service will attempt to cancel all queued or executing command requests associated with the client connection on which this command is received.

Property value constraints:

```
uniqueItems: true
minItems: 1
minimum: 1
```

Completion Message

Payload	Type	Required
`{`		
`"errorCode": "noMatchingRequestIDs",`	string	
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	string	
`}`		

Properties

errorCode

Specifies the error code if applicable. The following values are possible:

- `noMatchingRequestIDs` - No queued or executing command matches the requestIDs property.

completionCode

The completion code.

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

Properties
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

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

3.2.5 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 canceled.
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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

- Common.PowerSaveChangeEvent

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

3.2.6 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 application 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 timeout but can be canceled.
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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

Event Messages

None

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

3.2.7 Common.SetTransactionState

This command allows the application 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 an application 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 The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

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

3.2.8 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 canceled.
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 an application 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

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

3.2.9 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 device 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 canceled.
default: 0

Completion Message

Payload	Type	Required
`{`		
`"commandNonce": "646169ECDD0E440C2CECC8DDD7C27C22",`	string	
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	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 an integer string, or HEX (upper case.)
Property value constraints:
`pattern: ^[0-9A-F]{32}$|^[0-9]*$`

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

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

3.2.10 Common.ClearCommandNonce

Clear the command nonce from the device. The command nonce is included in an Authorisation Token for a command that will be used to ensure end to end security.

Clearing this value from the device will make any tokens with the old nonce invalid. It will not be possible to use any token, or perform any end to end secured operation, until a new nonce is created with Common.GetCommandNonce and a new token is created.

There is no requirement for the client to clear the command nonce, but doing so may be useful for various reasons:

1. Clearing the command nonce once the application has finished with it will stop an attacker from using that value and may help improve security.
2. Clearing the command nonce will cause the Common.NonceClearedEvent event to be fired immediately which avoids the client having to handle it at a later time. This could make event handling simpler.

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 canceled.
default: 0

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

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

3.3 Unsolicited Messages

3.3.1 Common.PowerSaveChangeEvent

This event specifies that the power save recovery time has changed.

Payload	Type	Required
`{`		
`"powerSaveRecoveryTime": 0`	integer	
`}`		

Properties

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.

Property value constraints:

```
minimum: 0
```

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

3.3.2 Common.DevicePositionEvent

This event reports that the device has changed its position status.

Payload	Type	Required
`{`		
`  "position": "inPosition"`	string	
`}`		

Properties

position

Provides the position of the device as one of the following values:

- `inPosition` - The device is in its normal operating position.
- `notInPosition` - The device has been removed from its normal operating position.
- `unknown` - The position of the device cannot be determined.

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

3.3.3 Common.NonceClearedEvent

This event reports that the end to end security nonce value has been cleared on the device. This could be because the nonce was explicitly cleared with Common.ClearCommandNonce, automatically cleared by a timeout, or cleared by actions documented for each device.

Payload	Type	Required
`{`		
`"reasonDescription": "Nonce cleared by timeout"`	string	
`}`		

Properties
reasonDescription optional text describing why the nonce was cleared. The value of this text shouldn't be relied on.

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

3.3.4 Common.ExchangeStateChangedEvent

This event reports that the service's exchange state has changed.

Payload	Type	Required
`{`		
`"exchange": "notSupported"`	string	
`}`		

Properties

exchange

Specifies the exchange state of the service. Exchange can used to perform a manual replenishment of a device and is entered by Storage.StartExchange and completed by Storage.EndExchange. When the state changes a Common.ExchangeStateChangedEvent is posted. Following values are possible:

- `notSupported` - Exchange is not supported on this service.
- `active` - Exchange is active on this service. Commands which interact with the device may be rejected with an error code as appropriate.
- `inactive` - Exchange is not active on this service.

default: "notSupported"

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

4. Card Reader Interface

This chapter defines the Card Reader interface functionality and messages.

4.1 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 storage units from which cards can be dispensed. Some have storage units in which a card can temporarily be parked to enable another card to be moved into the card reader.

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

In addition to the pure reading of the tracks mentioned above, security boxes can be used via this service to check the data of writable tracks for manipulation. These boxes (such as CIM or MM) are sensor-equipped devices that are able to check some other information on the card and compare it with the track data.

When the service controls a permanently connected chip card, unsupportedCommand will be returned to all commands except Common.Status, Common.Capabilities, CardReader.ChipPower, CardReader.ChipIO and CardReader.Reset.

The following defines the roles and responsibilities of an application within EMV: A distinction needs to be made between EMV Contact support and EMV Contactless support.

When defining an EMV Contact implementation:

- EMV Level 2 interaction is handled by the client or above.
- EMV Level 1 interaction is handled by the device.

All EMV status information that is defined as a Level 1 responsibility in the EMV specification should be handled by the service.

EMVCo grants EMV Level 1 Approvals to contact IFMs and EMVCo Level 2 Approvals to Application Kernels.

When defining an EMV Contactless implementation, the responsibilities will depend on the type of EMV contactless product being implemented.

There are different EMVCo defined product types. They can be found in the EMVCo Type Approval – Contactless Product – Administrative Process document [Ref. cardreader-1]). In this specification when referring to the Contactless Product Type, Intelligent Card Reader, the following must be included and handled by the device:

- An EMVCo Approved Level 1 Contactless PCD
- Entry Point and POS System Architecture according to Book A and B
- EMV Kernels according to Book C1 to C7 (minimum one kernel needs to be supported)

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

The Network, Consumer and Merchant Interfaces will be managed by the client or above.

4.2 General Information

4.2.1 References

ID	Description
cardreader-1	EMVCo Terminal Type Approval Contactless Product Administrative Process 2.9
cardreader-2	EMVCo Integrated Circuit Card Specifications for Payment Systems Version 4.3
cardreader-3	EMVCo Contactless Specifications for Payment Systems, Version 2.4

4.2.2 Intelligent Contactless Card Reader

In relation to contactless transactions, the terminology used in this specification is based on the EMV Contactless Specifications for Payment Systems. See References.

There are a number of types of payment systems (or EMV) compliant contactless card readers, from the Intelligent Card Reader (IDC); where the reader device handles most of the transaction processing and only returns the result, to a transparent card reader; where the contactless card reader device provides a generic communication channel to the card without having any in-built transaction processing capabilities.

A contactless payment system transaction can be performed in two different ways, magnetic stripe emulation; where the data returned from the chip is formatted as if it was read from the magnetic stripe, and EMV-like; where, in a similar way to a contact EMV transaction, the chip returns a full set of BER-TLV (Basic Encoding Rules-Tag Length Value) data. Each payment system defines when each type, or profile, is used for a transaction, but it is usually dependent on both the configuration of the terminal and contactless card being tapped.

This specification will use “magnetic stripe emulation” and “EMV-like” to identify the two profiles of contactless transactions.

Support for a generic contactless communication channel to the card is provided via the CardReader.ChipIO command. This is suitable for use with a transparent contactless card reader or with an intelligent contactless card reader device operating in a pass through mode.

The CardReader.ReadRawData command can be used with an intelligent contactless card reader device to provide magnetic track emulation transactions. Only magnetic track emulation transactions can be supported using this command.

When using an intelligent contactless card reader to support both EMV-like and magnetic track emulation transactions a number of commands are required. The CardReader.EMVClessConfigure command allows the exchange of data to configure the reader for card acceptance and the CardReader.EMVClessPerformTransaction command enables the reader and performs the transaction with the card when it is tapped. In most cases all the transaction steps involving the card are completed within the initial card tap. A sequence diagram showing the expected IDC command sequences, as well as the cardholder and application actions when performing a contactless card based transaction.

Some contactless payment systems allow a 2nd tap of the contactless card. For example a 2nd tap can be used to process authorization data received from the host. In the case of issuer update data this second tap is performed via the CardReader.EMVClessIssuerUpdate command. A sequence diagram showing the expected CardReader command sequences, as well as the cardholder and application actions. The CardReader.EMVClessQueryApplications and *CardReader.EMVClessConfigure* commands specified later in this document refer to the EMV terminology “Application Identifier (AID) - Kernel Combinations”. A detailed explanation can be found in Refs. cardreader-2 and cardreader-3.

This document refers to BER-TLV tags. These are defined by each individual payment systems and contain the data exchanged between the application, contactless card and an intelligent contactless card reader. They are used to

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

configure and prepare the intelligent contactless card reader for a transaction and are also part of the data that is returned by the reader on completion of the cards tap.

Based on the applicable payment system the application is expected to know which tags are required to be configured, what values to use for the tags and how to interpret the tags returned. Intelligent readers are expected to know the BER-TLV tag definitions supported per payment system application. The tags provided in this document are examples of the types of tags applicable to each command. They are not intended to be a definite list.

Intelligent Contactless Card Reader Sequence Diagrams

This section illustrates the sequence diagrams of EMV-like contactless intelligent card reader transactions.

Single Tap Transaction Without Issuer Update Processing

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

Double Tap Transaction With Issuer Update Processing

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

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

Card Removed Before Completion

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

4.3 Command Messages

4.3.1 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 canceled.
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

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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.

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

Event Messages

None

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

4.3.2 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	
`}`		

Properties

timeout
Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be canceled.
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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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.

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

Event Messages

None

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

4.3.3 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.Move 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": false,`	boolean	
`"track2": false,`	boolean	
`"track3": false,`	boolean	
`"chip": false,`	boolean	
`"security": false,`	boolean	
`"fluxInactive": false,`	boolean	
`"watermark": false,`	boolean	
`"memoryChip": false,`	boolean	
`"track1Front": false,`	boolean	
`"frontImage": false,`	boolean	
`"backImage": false,`	boolean	
`"track1JIS": false,`	boolean	
`"track3JIS": false,`	boolean	
`"ddi": false`	boolean	
`}`		

© 2021 CEN 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 canceled. 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

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

Properties
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.`		
`},`		

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

Payload	Type	Required
"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

The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription

If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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.

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

Properties
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.

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

Event Messages

- CardReader.InsertCardEvent
- CardReader.MediaInsertedEvent
- CardReader.MediaRemovedEvent
- CardReader.InvalidMediaEvent
- CardReader.TrackDetectedEvent

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

4.3.4 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": "QmFzZTY0IGVuY29kZWQgZGF0YQ==",`	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 canceled.
default: 0

data
An array of card data structures

data/
A card data structure

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

Properties
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 The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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

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

4.3.5 CardReader.Move

This command is only applicable to motorized and latched dip card readers.

If after a successful completion event the card is at the exit position, the card will be accessible to the user. A CardReader.MediaRemovedEvent is generated to inform the application when the card is taken.

Motorized card readers

Motorized card readers can physically move cards from or to the transport or exit positions or a storage unit. The default operation is to move a card in the transport position to the exit position.

If the card is being moved from the exit position to the exit position, these are valid behaviors:

1. The card does not move as the card reader can detect the card is already in the correct position.
2. The card is moved back into the card reader then moved back to the exit to ensure the card is in the correct position.

Latched dip card readers

Latched dips card readers can logically move cards from the transport position to the exit position by unlatching the card reader. That is, the card will not physically move but will be accessible to the user.

Command Message

Payload	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"from": "unit1",`	string	
`"to": "exit"`	string	
`}`		

Properties

timeout
Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be canceled.
default: 0

from
Specifies where the card should be moved from as one of the following:

- `exit` - The card will be moved from the exit position.
- `transport` - The card will be moved from the transport position. This is the only value applicable to latched dip card readers.
- `<storage unit identifier>` - The card will be moved from the storage unit with matching identifier. The storage unit type must be either *dispense* or *park*.

Property value constraints:

```
pattern: ^exit$|^transport$|^.{1,5}$
```

default: "transport"

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

Properties
to Specifies where the card should be moved to as one of the following: • `exit` - The card will be moved to the exit. This is the only value applicable to latched dip card readers. • `transport` - The card will be moved to the transport just behind the exit slot. • `<storage unit identifier>` - The card will be moved to the storage unit with matching identifier. The storage unit type must be either *retain* or *park*. Property value constraints: `pattern: ^exit$\|^transport$|^.{1,5}$` default: "exit"

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "mediaJam"`	string	
`}`		

Properties
completionCode The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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` - No card is in the requested *from* position. • `occupied` - A card already occupies the requested *to* position. • `mediaRetained` - The card has been retained during attempts to move it to the exit position. The device is clear and can be used.

Event Messages

- CardReader.MediaRemovedEvent
- Storage.StorageThresholdEvent

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

4.3.6 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 canceled.
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
The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

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

4.3.7 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 canceled. 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	

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

Payload	Type	Required
`}`		

Properties

completionCode

The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription

If included, this contains additional vendor dependent information to assist with problem resolution.

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

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

4.3.8 CardReader.Reset

This command is used by the client to perform a hardware reset which will attempt to return the card reader device to a known good state.

If the device is a user card reader:

- Dependent on the command parameters, the device will attempt to move a card in the exit or transport positions to the exit or transport positions or a retain storage unit.
- For each user card in the device (including parking storage units), a CardReader.MediaDetectedEvent will inform the application where the card is on completion of this command.
- It may not always be possible to move the cards because of hardware problems.

If the device is a permanent chip card unit, this command will power-off the chip.

Command Message

Payload	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"to": "retain",`	string	
`"storageId": "retn1"`	string	
`}`		

Properties

timeout

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

default: 0

to

Specifies where any user card found within the card reader should be moved to as one of the following:

- `exit` - Move the card to the exit position. If the card is already at the exit, it may be moved to ensure it is in the correct position to be taken.
- `retain` - Move the card to a retain storage unit.
- `transport` - Move the card to the transport or keep the card in the transport. If the card is already in the transport, it may be moved to verify it is not jammed.

storageId

If the card is to be moved to a retain storage unit, this indicates the retain storage unit to which the card should be be moved. If omitted, the service will select the retain storage unit based on the number of retain storage units available and service specific configuration.

Property value constraints:

```
pattern: ^.{1,5}$
```

Completion Message

This event notifies the completion of the command and if successful includes the requested data.

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	

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

Payload	Type	Required
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "mediaJam"`	string	
`}`		

Properties

completionCode

The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription

If included, this contains additional vendor dependent information to assist with problem resolution.

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
- CardReader.MediaDetectedEvent
- Storage.StorageThresholdEvent

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

4.3.9 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 canceled.
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
The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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

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

4.3.10 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 canceled.

default: 0

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

Properties
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 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.

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

Properties
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 The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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

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

4.3.11 CardReader.EMVClessPerformTransaction

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 canceled.

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.

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

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": {	object	
See uiOutcome properties.		
},		
"fieldOffHoldTime": 0,	integer	
"cardRemovalTimeout": 0,	integer	
"discretionaryData": Add example to YAML	string	
}		
},		
"track2": {	object	
See track1 properties.		
},		
"track3": {	object	
See track1 properties.		
},		
"chip": {	object	

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

Payload	Type	Required
`See track1 properties.`		
`}`		
`}`		

Properties

completionCode

The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription

If included, this contains additional vendor dependent information to assist with problem resolution.

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.EMVClessConfigure 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.EMVClessIssuerUpdate 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.EMVClessPerformTransaction 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).

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

Properties
track1/cardholderAction Specifies the card holder 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 card holder verification method (CVM) to be performed as one of the following: • `onlinePIN` - Online PIN should be entered by the card holder. • `confirmationCodeVerified` - A confirmation code entry has been successfully done on a mobile device. • `sign` - Application should obtain card holder 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 card holder 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).

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

Properties
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 card holder 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.

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

Properties
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

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

4.3.12 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 WFSExecute call for a card to be tapped.

Command Message

Payload	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"data": Add example to YAML`	string	Yes
`}`		

Properties

timeout

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

default: 0

data

Base64 encoded representation of the EMV data elements in a BER-TLV format received 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	
`"clessOutcome": {`	object	
`"cvm": "onlinePIN",`	string	

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

Payload	Type	Required
`"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

The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription

If included, this contains additional vendor dependent information to assist with problem resolution.

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.

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

Properties
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.EMVClessIssuerUpdate 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.EMVClessPerformTransaction 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 card holder 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 card holder verification method (CVM) to be performed as one of the following: • `onlinePIN` - Online PIN should be entered by the card holder. • `confirmationCodeVerified` - A confirmation code entry has been successfully done on a mobile device. • `sign` - Application should obtain card holder 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.

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

Properties
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 card holder 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: ""

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

Properties
chip/clessOutcome/uiRestart The user interface details required to be displayed to the card holder 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

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

4.4 Event Messages

4.4.1 CardReader.InsertCardEvent

This event notifies the application when the device is ready for the user to insert a card.

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

4.4.2 CardReader.MediaInsertedEvent

This event notifies the application when the device is ready for the user to insert a card.

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

4.4.3 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.

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

4.4.4 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

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

4.4.5 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.

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

4.4.6 Storage.StorageThresholdEvent

This user event is generated when a threshold condition has occurred in one of the storage units.

This event can be triggered either by hardware sensors in the device or by count thresholds being met.

Payload	Type	Required

{
  "storage": {
  "unit1": {
  "positionName": "Top Right",
  "capacity": 100,
  "status": "ok",
  "serialNumber": "ABCD1234",
  "cash": {
  "capabilities": {
  "types": {
  "cashIn": false,
  "cashOut": false,
  "replenishment": false,
  "cashInRetract": false,
  "cashOutRetract": false,
  "reject": false
  },
  "items": {
  "fit": false,
  "unfit": false,
  "unrecognized": false,
  "counterfeit": false,
  "suspect": false,
  "inked": false,
  "coupon": false,
  "document": false
  },
  "hardwareSensors": false,
  "retractAreas": 1,
  "retractThresholds": false,
  "cashItems": {
  "20USD1": {
  "noteID": 1,		

	object	
	object	
	string	
	integer	
	string	Yes
	string	
	object	
	object	
	object	
	object	
	boolean	
	boolean	
	boolean	
	boolean	
	boolean	
	boolean	
	object	
	boolean	
	boolean	
	boolean	
	boolean	
	boolean	
	boolean	
	boolean	
	boolean	
	boolean	
	integer	
	boolean	
	object	
	object	
	integer	

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

Payload	Type	Required
"currency": "USD",	string	
"value": 20,	number	
"release": 1	integer	
}		
}		
},		
"configuration": {	object	
"types": {	object	
See types properties.		
},		
"items": {	object	
See items properties.		
},		
"currency": "USD",	string	
"value": 20,	number	
"highThreshold": 500,	integer	
"lowThreshold": 10,	integer	
"appLockIn": false,	boolean	
"appLockOut": false,	boolean	
"cashItems": {	object	
See cashItems properties.		
},		
"name": "$10",	string	
"maxRetracts": 5	integer	
},		
"status": {	object	
"index": 1,	integer	
"initial": {	object	
"unrecognized": 0,	integer	
"cash": {	object	
"*20USD1*": {	object	
"fit": 0,	integer	
"unfit": 0,	integer	
"suspect": 0,	integer	
"counterfeit": 0,	integer	
"inked": 0	integer	
}		

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

Payload	Type	Required
}		
},		
"out": {	object	
"presented": {	object	
See initial properties.		
},		
"rejected": {	object	
See initial properties.		
},		
"distributed": {	object	
See initial properties.		
},		
"unknown": {	object	
See initial properties.		
},		
"stacked": {	object	
See initial properties.		
},		
"diverted": {	object	
See initial properties.		
},		
"transport": {	object	
See initial properties.		
}		
},		
"in": {	object	
"retractOperations": 0,	integer	
"deposited": {	object	
See initial properties.		
},		
"retracted": {	object	
See initial properties.		
},		
"rejected": {	object	
See initial properties.		
},		
"distributed": {	object	

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

Payload	Type	Required
See initial properties.		
}		
},		
"count": 0,	integer	
"accuracy": "notSupported",	string	
"replenishmentStatus": "ok"	string	
}		
},		
"card": {	object	
"capabilities": {	object	
"type": "retain",	string	
"hardwareSensors": false	boolean	
},		
"configuration": {	object	
"cardID": "LoyaltyCard",	string	
"threshold": 0	integer	
},		
"status": {	object	
"initialCount": 0,	integer	
"count": 0,	integer	
"retainCount": 0,	integer	
"replenishmentStatus": "ok"	string	
}		
}		
},		
"unit2": {	object	
See unit1 properties.		
}		
}		
}		

Properties

storage
Object containing storage unit information. The property name is the storage unit identifier.

storage/unit1 (example name)
The object contains a single storage unit.
Property name constraints:
`pattern: ^.{1,5}$`

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

Properties
storage/unit1/positionName Fixed physical name for the position.
storage/unit1/capacity The nominal capacity of the unit. This may be an estimate as the quality and thickness of the items stored in the unit may affect how many items can be stored. 0 means the capacity is unknown. Property value constraints: `minimum: 0`
storage/unit1/status The state of the unit. The following values are possible: • `ok` - The storage unit is in a good state. • `inoperative` - The storage unit is inoperative. • `missing` - The storage unit is missing. • `notConfigured` - The storage unit has not been configured for use. • `manipulated` - The storage unit has been inserted (including removal followed by a reinsertion) when the device was not in the exchange state - see Storage.StartExchange. This storage unit cannot be used. Only applies to services which support the exchange state.
storage/unit1/serialNumber The storage unit's serial number if it can be read electronically.
storage/unit1/cash The cash related contents, status and configuration of the unit.
storage/unit1/cash/capabilities Indicates what the storage unit is capable of - this includes information which is a combination of that reported in WFS_INF_CDM_CASH_UNIT_INFO, WFS_INF_CIM_CASH_UNIT_INFO and WFS_INF_CIM_CASH_UNIT_CAPABILITIES in XFS 3.x.
storage/unit1/cash/capabilities/types The types of operation the unit is capable of or configured to perform. This is a combination of one or more operations. May only be modified in an exchange state if applicable.
storage/unit1/cash/capabilities/types/cashIn The unit can accept cash items. If *cashOut* is also true then the unit can recycle.
storage/unit1/cash/capabilities/types/cashOut The unit can dispense cash items. If *cashIn* is also true then the unit can recycle.
storage/unit1/cash/capabilities/types/replenishment Replenishment container. A storage unit can be refilled from or emptied to a replenishment container.
storage/unit1/cash/capabilities/types/cashInRetract Retract unit. Items can be retracted into this unit during Cash In operations.
storage/unit1/cash/capabilities/types/cashOutRetract Retract unit. Items can be retracted into this unit during Cash Out operations.
storage/unit1/cash/capabilities/types/reject Reject unit. Items can be rejected into this unit.
storage/unit1/cash/capabilities/items The types of cash media the unit is capable of storing or configured to store. This is a combination of one or more item types. May only be modified in an exchange state if applicable. See Note Classification for more information about cash classification levels.

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

Properties
storage/unit1/cash/capabilities/items/fit The storage unit can store cash items which are fit for recycling (classification level 4).
storage/unit1/cash/capabilities/items/unfit The storage unit can store cash items which are unfit for recycling (classification level 4).
storage/unit1/cash/capabilities/items/unrecognized The storage unit can store unrecognized cash items (classification level 1).
storage/unit1/cash/capabilities/items/counterfeit The storage unit can store counterfeit cash items (classification level 2).
storage/unit1/cash/capabilities/items/suspect The storage unit can store suspect cash items (classification level 3).
storage/unit1/cash/capabilities/items/inked The storage unit can store cash items which have been identified as ink stained.
storage/unit1/cash/capabilities/items/coupon Storage unit containing coupons or advertising material.
storage/unit1/cash/capabilities/items/document Storage unit containing documents.
storage/unit1/cash/capabilities/hardwareSensors Indicates whether the storage unit has sensors which report the status.
storage/unit1/cash/capabilities/retractAreas If items can be retracted into this storage unit, this is the number of areas within the storage unit which allow physical separation of different bunches. If there is no physical separation of retracted bunches within this storage unit, this value is 1. Property value constraints: `minimum: 1`
storage/unit1/cash/capabilities/retractThresholds If true, indicates that retract capacity is based on counts. If false, indicates that retract capacity is based on the number of commands which resulted in items being retracted into the storage unit.
storage/unit1/cash/capabilities/cashItems Lists the cash items which the storage unit is physically capable of handling. For example a coin storage unit may be restricted to one coin denomination due to the hardware.
storage/unit1/cash/capabilities/cashItems/20USD1 (example name) An object containing information about a single cash item supported by the device.
storage/unit1/cash/capabilities/cashItems/20USD1/noteID Assigned by the XFS4IoT service. A unique number identifying a single cash item. Each unique combination of the other properties will have a different noteID. Can be used for migration of *usNoteID* from XFS 3.x. Property value constraints: `minimum: 1`
storage/unit1/cash/capabilities/cashItems/20USD1/currency ISO 4217 currency. Property value constraints: `maxLength: 3`

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

Properties
storage/unit1/cash/capabilities/cashItems/20USD1/value Absolute value of all contents, 0 if mixed. May only be modified in an exchange state if applicable. May be a floating point value to allow for coins and notes which have a value which is not a whole multiple of the currency unit.
storage/unit1/cash/capabilities/cashItems/20USD1/release The release of the cash item. The higher this number is, the newer the release. If zero or not reported, there is only one release of that cash item or the device is not capable of distinguishing different release of the item, for example in a simple cash dispenser. An example of how this can be used is being able to sort different releases of the same denomination note to different storage units to take older notes out of circulation. This value is device, banknote reader and currency dependent, therefore a release number of the same cash item will not necessarily have the same value in different systems and any such usage would be specific to a specific device's configuration. Property value constraints: `minimum: 0`
storage/unit1/cash/configuration Indicates what this storage unit is configured as or is being configured to do - where applicable the supported options can be derived from capabilities. If the service supports an exchange state, only a subset of these parameters may be modified unless in an exchange. Parameters which may only be modified in an exchange state are listed.
storage/unit1/cash/configuration/types The types of operation the unit is capable of or configured to perform. This is a combination of one or more operations. May only be modified in an exchange state if applicable.
storage/unit1/cash/configuration/items The types of cash media the unit is capable of storing or configured to store. This is a combination of one or more item types. May only be modified in an exchange state if applicable. See Note Classification for more information about cash classification levels.
storage/unit1/cash/configuration/currency ISO 4217 currency. May only be modified in an exchange state if applicable. Property value constraints: `maxLength: 3`
storage/unit1/cash/configuration/highThreshold If specified, replenishmentStatus is set to *high* if count is greater than this number. If not specified, high is based on hardware sensors if supported - see hardwareSensors. Property value constraints: `minimum: 1`
storage/unit1/cash/configuration/lowThreshold If specified, replenishmentStatus is set to *low* if count is less than this number. If not specified, low is based on hardware sensors if supported - see hardwareSensors. Property value constraints: `minimum: 1`
storage/unit1/cash/configuration/appLockIn If true, items cannot be accepted into the storage unit in Cash In operations.
storage/unit1/cash/configuration/appLockOut If true, items cannot be dispensed from the storage unit in Cash Out operations.

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

Properties
storage/unit1/cash/configuration/cashItems An object containing multiple cash items, listing what a storage unit is capable of or configured to handle.
storage/unit1/cash/configuration/name Application configured name of the unit.
storage/unit1/cash/configuration/maxRetracts If specified, this is the number of retract operations allowed into the unit. If not specified, the maximum number is not limited by counts. Property value constraints: `minimum: 1`
storage/unit1/cash/status Indicates the storage unit status - this includes information which is a combination of that reported in WFS_INF_CDM_CASH_UNIT_INFO and WFS_INF_CIM_CASH_UNIT_INFO in XFS 3.x.
storage/unit1/cash/status/index Assigned by the XFS4IoT service. Will be a unique number which can be used to determine *usNumber* in XFS 3.x migration. This can change as cash storage units are added and removed from the storage collection. Property value constraints: `minimum: 1`
storage/unit1/cash/status/initial The cash related items which were in the storage unit at the last replenishment.
storage/unit1/cash/status/initial/unrecognized Count of unrecognized items handled by the cash interface
storage/unit1/cash/status/initial/cash Counts of cash items broken down by cash item type and classification
storage/unit1/cash/status/initial/cash/20USD1 (example name) Counts of a given CashItem broken down by classification.
storage/unit1/cash/status/initial/cash/20USD1/fit Count of genuine cash items which are fit for recycling (classification level 4 - see Note Classification).
storage/unit1/cash/status/initial/cash/20USD1/unfit Count of genuine cash items which are unfit for recycling (classification level 4 - see Note Classification).
storage/unit1/cash/status/initial/cash/20USD1/suspect Count of suspect cash items (classification level 3 - see Note Classification).
storage/unit1/cash/status/initial/cash/20USD1/counterfeit Count of counterfeit cash items (classification level 2 - see Note Classification).
storage/unit1/cash/status/initial/cash/20USD1/inked Count of cash items which have been identified as ink stained.
storage/unit1/cash/status/out The items moved from this storage unit by cash commands to another destination since the last replenishment of this unit. Reset to empty if initial is set for this unit by Storage.GetStorage.
storage/unit1/cash/status/out/presented The items dispensed from this storage unit which are or were customer accessible.

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

Properties
storage/unit1/cash/status/out/rejected The items dispensed from this storage unit which were invalid and were diverted to a reject storage unit and were not customer accessible during the operation.
storage/unit1/cash/status/out/distributed The items dispensed from this storage unit which were moved to a storage unit other than a reject storage unit and were not customer accessible during the operation.
storage/unit1/cash/status/out/unknown The items dispensed from this storage unit which moved to an unknown position.
storage/unit1/cash/status/out/stacked The items dispensed from this storage unit which are not customer accessible and are currently stacked awaiting presentation to the customer. This item list can increase and decrease as items are moved around in the device.
storage/unit1/cash/status/out/diverted The items dispensed from this storage unit which are not customer accessible and were diverted to a temporary location due to being invalid and have not yet been deposited in a storage unit. This item list can increase and decrease as items are moved around in the device.
storage/unit1/cash/status/out/transport The items dispensed from this storage unit which are not customer accessible and which have jammed in the transport. This item list can increase and decrease as items are moved around in the device.
storage/unit1/cash/status/in List of items inserted in this storage unit by cash commands from another source since the last replenishment of this unit. Reset to empty if initial is set for this unit by Storage.GetStorage.
storage/unit1/cash/status/in/retractOperations Number of cash retract operations which resulted in items entering this storage unit. This can be used where devices do not have the capability to count or validate items after presentation.
storage/unit1/cash/status/in/deposited The items deposited in the storage unit during a Cash In transaction.
storage/unit1/cash/status/in/retracted The items deposited in the storage unit after being accessible to a customer. This may be inaccurate or not counted if items are not counted or re-validated after presentation, the number of retract operations is also reported separately in *retractOperations*.
storage/unit1/cash/status/in/rejected The items deposited in this storage unit originating from another storage unit but rejected due to being invalid. This count may be inaccurate due to the nature of rejected items.
storage/unit1/cash/status/in/distributed The items deposited in this storage unit originating from another storage unit but not rejected.

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

Properties
storage/unit1/cash/status/count Total count of the items in the unit, derived from the initial, out and in counts, but supplied for ease of use. For units which dispense items, this count is only decremented when the items are either known to be in customer access or successfully rejected, therefore the intermediate *out* fields are not included in this calculation: • stacked • transport • unknown If counts being incorrectly set at replenishment time means that this would result in a negative number, this reports 0. Property value constraints: `minimum: 0`
storage/unit1/cash/status/accuracy Describes the accuracy of *count*. Following values are possible: • `notSupported` - The hardware is not capable of determining the accuracy of *count*. • `accurate` - The *count* is expected to be accurate. The notes were previously counted and there have since been no events that might have introduced inaccuracy. • `accurateSet` - The *count* is expected to be accurate. The counts 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 storage unit insertion or some other hardware event.
storage/unit1/cash/status/replenishmentStatus The state of the media in the unit if it can be determined. Note that overall status of the storage unit must be taken into account when deciding whether the storage unit is usable and whether replenishment status is applicable. In particular, if the overall status is *missing* this will not be reported. The following values are possible: • `ok` - The storage unit media is in a good state. • `full` - The storage unit is full. • `high` - The storage unit is almost full (either sensor based or exceeded the highThreshold. • `low` - The storage unit is almost empty (either sensor based or below the lowThreshold). • `empty` - The storage unit is empty, or insufficient items in the storage unit are preventing further dispense operations.
storage/unit1/card The card related contents, status and configuration of the unit.
storage/unit1/card/capabilities Indicates the card storage unit capabilities.
storage/unit1/card/capabilities/type The type of card storage as one of the following: • `retain` - The storage unit can retain cards. • `dispense` - The storage unit can dispense cards. • `park` - The storage unit can be used to temporarily store a card allowing another card to enter the transport.
storage/unit1/card/capabilities/hardwareSensors The storage unit has hardware sensors that can detect threshold states. default: false

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

Properties
storage/unit1/card/configuration Indicates the card storage unit configuration.
storage/unit1/card/configuration/cardID The identifier that may be used to identify the type of cards in the storage unit. This is only applicable to *dispense* type storage units.
storage/unit1/card/configuration/threshold If the threshold value is non zero, hardware sensors in the storage unit do not trigger Storage.StorageThresholdEvent events. If non zero, when *count* reaches the threshold value: • For retain type storage units, a high threshold will be sent. • For dispense type storage units, a low threshold will be sent. Property value constraints: `minimum: 0`
storage/unit1/card/status Indicates the card storage unit status.
storage/unit1/card/status/initialCount The initial number of cards in the storage unit. This is only applicable to dispense type storage units. This value is persistent. Property value constraints: `minimum: 0`
storage/unit1/card/status/count The number of cards in the storage unit. If the storage unit type is dispense: • This count also includes a card dispensed from the storage unit which has not been moved to either the exit position or a dispense type storage unit. • This count is decremented when a card from the card storage unit is moved to the exit position or retained. If this value reaches zero it will not decrement further but will remain at zero. If the storage unit type is retain: • The count is incremented when a card is moved into the storage unit. If the storage unit type is park: • The count will increment when a card is moved into the storage module and decremented when a card is moved out of the storage module. This value is persistent. Property value constraints: `minimum: 0`
storage/unit1/card/status/retainCount The number of cards from this storage unit which are in a retain storage unit. This is only applicable to dispense type storage units. This value is persistent. Property value constraints: `minimum: 0`

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

Properties
storage/unit1/card/status/replenishmentStatus The state of the cards in the storage unit if it can be determined. Note that overall status of the storage unit must be taken into account when deciding whether the storage unit is usable and whether replenishment status is applicable. In particular, if the overall status is *missing* this will be omitted. The following values are possible: • `ok` - The storage unit is in a good state. • `full` - The storage unit is full. • `high` - The storage unit is almost full (either sensor based or above the threshold). • `low` - The storage unit is almost empty (either sensor based or below the threshold). • `empty` - The storage unit is empty.

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

4.4.7 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 park storage units, there will be one event for each card found.

Payload	Type	Required
`{`		
`"position": "retn1"`	string	
`}`		

Properties

position

Specifies where the card was moved to or if it is jammed as one of the following:

- `exit` - The card was moved to the exit position.
- `transport` - The card was moved to the transport position.
- `<storage unit identifier>` - The card was moved to the storage unit with matching identifier. The storage unit type must be either *retain*.
- `jammed` - The card is jammed in the device.

Property value constraints:

```
pattern: ^exit$|^transport$|^.{1,5}$
```

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

4.4.8 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.

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

Properties
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: ""

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

4.5 Unsolicited Messages

4.5.1 CardReader.MediaRemovedEvent

This unsolicited event indicates the card was manually removed by the user either during processing of a command which requires the card to be present or the card is removed from the exit position.

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

4.5.2 CardReader.CardActionEvent

This event specifies where a card has been moved to by either the automatic power on or power off action of the device.

Payload	Type	Required
`{`		
`"to": "retn1",`	string	
`"from": "transport"`	string	
`}`		

Properties

to

Position where the card was moved to. Possible values are:

- `exit` - The card was moved to the exit position.
- `transport` - The card was moved to the transport position.
- `<storage unit identifier>` - The card was moved to the storage unit with matching identifier. The storage unit type must be *retain*.

Property value constraints:

```
pattern: ^exit$|^transport$|^.{1,5}$
```

from

Position where the card was moved from. Possible values are:

- `unknown` - The position of the card cannot be determined.
- `exit` - The card was in the exit position.
- `transport` - The card was moved in the transport position.
- `<storage unit identifier>` - The card was in a storage unit with matching identifier. The storage unit type must be *park*.

Property value constraints:

```
pattern: ^unknown$|^exit$|^transport$|^.{1,5}$
```

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

5. Cash Management Interface

This chapter defines the Cash Management interface functionality and messages.

5.1 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.

5.2 General Information

5.2.1 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.

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

For example, "S8H9??16?4" would represent a match for the serial numbers "S8H9231654" and "S8H9761684". A mask of "HD902" 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".

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

5.3 Command Messages

5.3.1 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 canceled.
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	

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

Payload	Type	Required
"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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

cashunits
Object containing cash unit information.

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

Properties
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 application 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 application to assign a value to this field. This value is persistent.

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

Properties
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 application 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 canceled 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.

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

Properties
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.

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

Properties
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

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

5.3.2 CashManagement.GetTellerInfo

This command only applies to Teller devices. It allows the application 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 application 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 canceled. 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	

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

Payload	Type	Required
`"cashBoxReceived": 0,`	number	
`"cashBoxDispensed": 0`	number	
`},`		
`}`		
`}]`		
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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.

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

Properties
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

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

5.3.3 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. CashDispenser.Dispense, CashDispenser.Count, CashDispenser.Present, CashDispenser.Retract, CashDispenser.Reject, CashDispenser.OpenShutter, CashDispenser.CloseShutter, CashDispenser.Reset, CashManagement.StartExchange, CashManagement.EndExchange, CashManagement.CalibrateCashUnit, CashDispenser.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 canceled.

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.

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

Properties
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)	
`"currencyID": Add example to YAML,`	string	
`"value": 0,`	number	
`"release": 0,`	integer	
`"noteID": 0,`	integer	
`"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	
`}]`		
`}`		

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

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
itemsList Array of "item info" objects.
itemsList/currencyID Currency ID in ISO 4217 format [Ref. 2]. This value will be omitted for level 1 items.
itemsList/value The value of a single item expressed as floating point value. This value will be zero for level 1 items.
itemsList/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. This value will be zero for level 1 items.
itemsList/noteID Identification of note type. This value will be zero for level 1 items.
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.

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

Properties
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

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

5.3.4 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 canceled.
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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

version
This is an application 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.

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

Properties
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

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

5.3.5 CashManagement.SetTellerInfo

This command allows the application 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 canceled.

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.

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

Properties
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

Payload	Type	Required
`{`		
`  "completionCode": "success",`	string	

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

Payload	Type	Required
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "invalidCurrency"`	string	
`}`		

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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

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

5.3.6 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 omitted.

This command generates the CashManagement.CashUnitInfoChangedEvent to inform applications that cash unit information has been changed.

This command can be used to change software counters, thresholds and the application 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	

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

Payload	Type	Required
"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 canceled. default: 0
cashunits Object containing cash unit settings.
cashunits/additionalProperties Cash unit settings.

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

Properties
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 application 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 application 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 application 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.

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

Properties
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 canceled 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.

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

Properties
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.

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

Properties
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 The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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

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

5.3.7 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 canceled.
default: 0

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "exchangeActive"`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

errorCode
Specifies the error code if applicable. Following values are possible:

- `exchangeActive` - The device is in an exchange state.

Event Messages

None

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

5.3.8 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 an application 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, CashDispenser.SetMixTable and Reset commands (e.g. CashDispenser.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	
`  }`		
`}`		

© 2021 CEN 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 canceled. 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.

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

Completion Message

Payload	Type	Required

{
  "completionCode": "success",
  "errorDescription": Add example to YAML,
  "errorCode": "invalidCurrency",
  "cashunits": {
  "additionalProperties": {
  "status": "ok",
  "type": "billCassette",
  "currencyID": Add example to YAML,
  "value": 0,
  "logicalCount": 0,
  "maximum": 0,
  "appLock": false,
  "cashUnitName": Add example to YAML,
  "initialCount": 0,
  "dispensedCount": 0,
  "presentedCount": 0,
  "retractedCount": 0,
  "rejectCount": 0,
  "minimum": 0,
  "physicalPositionName": Add example to YAML,
  "unitID": Add example to YAML,
  "count": 0,
  "maximumCapacity": 0,
  "hardwareSensor": false,
  "itemType": {
  "all": false,
  "unfit": false,
  "individual": false,
  "level1": false,
  "level2": false,
  "level3": false,
  "itemProcessor": false,
  "unfitIndividual": false
  },
  "cashInCount": 0,
	
	string
	string
	string
	object
	object
	string
	string
	string
	number
	integer
	integer
	boolean
	string
	integer
	integer
	integer
	integer
	integer
	integer
	string
	string
	integer
	integer
	boolean
	object
	boolean
	boolean
	boolean
	boolean
	boolean
	boolean
	boolean
	boolean
	
	integer	

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

Payload	Type	Required
`"noteNumberList": {`	object	
`"noteNumber": [{`	array (object)	
`"noteID": 0,`	integer	
`"count": 0`	integer	
`}]`		
`},`		
`"noteIDs": [0]`	array (integer)	
`},`		
`}`		
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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.

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

Properties
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 application 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 application 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 application 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.

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

Properties
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 canceled 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.

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

Properties
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.

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

Properties
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

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

5.3.9 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.

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 application 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 canceled.
default: 0

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "cashUnitError"`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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

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

5.3.10 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 application. 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 canceled.
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.

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

Properties
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	

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

Payload	Type	Required
`"position": {`	object	
`"cashunit": Add example to YAML,`	string	
`"retractArea": {`	object	
`"outputPosition": "default",`	string	
`"retractArea": "retract",`	string	
`"index": 0`	integer	
`},`		
`"outputPosition": "default"`	string	
`}`		
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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.

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

Properties
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
- CashDispenser.ItemsTakenEvent

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

5.3.11 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 canceled. default: 0
version This is an application 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.

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

Properties
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

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	string	
`}`		

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

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

5.4 Event Messages

5.4.1 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	
},		

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

Payload	Type	Required
`"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.

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

Properties
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 application 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 application 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 application 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.

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

Properties
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 canceled 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.

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

Properties
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.

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

Properties
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.

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

5.4.2 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:

- `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.

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

5.4.3 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 Number of items classified as *level* which have information available.

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

5.4.4 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.

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

5.4.5 CashDispenser.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.

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

5.5 Unsolicited Messages

5.5.1 CashManagement.SafeDoorOpenEvent

This event specifies that the safe door has been opened.

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

5.5.2 CashManagement.SafeDoorClosedEvent

This event specifies that the safe door has been closed.

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

5.5.3 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*):

CashDispenser.CalibrateCashUnit

CashDispenser.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 an application 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	

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

Payload	Type	Required
`"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.

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

Properties
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 application 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 application 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 application 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.

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

Properties
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 canceled 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.

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

Properties
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.

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

Properties
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.

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

5.5.4 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.

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

5.5.5 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 application 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	
`},`		

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

Payload	Type	Required
`"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 application to assign a value to this field. This value is persistent.

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

Properties
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 application 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 application 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.

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

Properties
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 canceled 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.

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

Properties
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.

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

6. Cash Dispenser Interface

This chapter defines the Cash Dispenser interface functionality and messages.

6.1 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.

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

6.2 Command Messages

6.2.1 CashDispenser.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 canceled.
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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

mixTypes
Array of mix type objects.

mixTypes/mixNumber
Number identifying the mix algorithm or the house mix table. This number can be passed to the CashDispenser.MixTable, CashDispenser.Dispense and CashDispenser.Denominate commands.

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

Properties
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. Application 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

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

6.2.2 CashDispenser.GetMixTable

This command is used to obtain the house mix table specified by the supplied mix number.

Command Message

Payload	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"mixNumber": 0`	integer	
`}`		

Properties

timeout
Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be canceled.
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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

errorCode
Specifies the error code if applicable. Following values are possible:

- `invalidMixNumber` - The *mixNumber* parameter does not correspond to a defined mix table.

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

Properties
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

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

6.2.3 CashDispenser.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 CashDispenser.Present or CashDispenser.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, including across power cycles.

If end to end security is supported then this value is *not* cleared if a dispense with an invalid token is received. If a dispense token is invalid the dispense will fail with an invalidToken error, and the GetPresentStatus command will continue to report the existing status. This is to stop an attacker being able to reset the present status and conceal the last present result.

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).

If end to end security is supported by the hardware, the present status will be protected by a security token. If end to end security is not supported then it's not possible to guarantee that the present status hasn't been altered, possibly by an attacker trying to hide the fact that cash was presented. To avoid this risk the client must always call GetPresentStatus and validate the security token.

If end to end security is being used the caller must pass in a nonce value. This value will be included in the security token that is returned. The caller must check that the original nonce value matches the token - if they do not match then the token is invalid. See the api

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 canceled.
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.

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

Properties
nonce A nonce value to be used when creating the end to end security token in the response. See the generic end to end security documentation for more details. Property value constraints: `pattern: ^[0-9A-F]{32}$\|^[0-9]*$`

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	

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

Payload	Type	Required
`},`		
`"cashBox": 0`	integer	
`},`		
`"presentState": "presented",`	string	
`"token": "NONCE=1414,TOKENFORMAT=1,TOKENLENGTH=0268, DISPENSEID=CB735612FD6141213C2827FB5A6A4F4846D7A7347B15434916FEA6AC16F3D2F2, DISPENSED1=50.00EUR,PRESENTED1=YES,PRESENTEDAMOUNT1=50.00EUR,RETRACTED1=NO, HMACSHA256=55D123E9EE64F0CC3D1CD4F953348B441E521BBACCD6998C6F51D645D71E6C83"`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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 CashDispenser.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 application 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.

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

Properties
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.
token The present status token that protects the present status. See the generic end to end document for more information.

Event Messages

None

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

6.2.4 CashDispenser.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 application.

This command can also be used to validate that any denomination supplied by the application 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. Alternativly the currency information can be omitted and the mix number must be 0 ("individual").

If the *cashBox* field returned by the CashDispenser.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 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	
`}`		
`}`		

© 2021 CEN 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 canceled. default: 0
tellerID Identification of teller. This field is ignored if the device is a Self-Service CashDispenser.
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 application 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	
`}`		

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

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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 application 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

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

6.2.5 CashDispenser.Dispense

This command performs the dispensing of items to the customer. The command provides the same functionality as the CashDispenser.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 CashDispenser.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 CashDispenser.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 CashDispenser, 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 CashDispenser.Present command to present the items to the user. If the device does not have an intermediate stacker the CashDispenser.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.

If the dispensed amount cannot be presented in one bunch of items, but the device can automatically split it into multiple bunches, this will be denoted by the *presentInBunches* field in the completion message. If it is set to "true" and the number of bunches is known the *bunches* field will also specify how many bunches are to be presented to the customer.

The process of dispensing and presenting cash may be protected by end to end security. This means that the hardware will generate a command nonce (returned by Common.GetCommandNonce) and the caller must use this to create a security token that authorizes dispensing the cash.

The command nonce, and any matching token, will stay valid until the cash is presented or might have become accessible in any other way. As soon as the cash has been presented the command nonce stored in the hardware will be cleared. This has the effect of making any existing tokens invalid so that they can't be used again.

The command nonce may be cleared for other reasons too, for example after a power failure or after a fixed time. Any tokens using the old command nonce value will become invalid when the command nonce is cleared.

© 2021 CEN 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	
"tellerID": 0,	integer	
"mixNumber": 0,	integer	
"position": "default",	string	
"denomination": {	object	
"currencies": {	object	
"additionalProperties": 0,	number	
},		
"values": {	object	
"additionalProperties": 0,	integer	
},		
"cashBox": 0	integer	
},		
"presentInBunches": "false",	string	
"bunches": 0,	integer	
"token": "NONCE=254611E63B2531576314E86527338D61, TOKENFORMAT=1,TOKENLENGTH=0164,DISPENSE1=50.00EUR, HMACSHA256=CB735612FD6141213C2827FB5A6A4F4846D7A7347B15434916FEA6AC16F3D2F2"	string	
}		

© 2021 CEN 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 canceled. default: 0
tellerID Identifies the teller. This field is ignored if the device is a Self-Service CashDispenser.
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 application 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.
presentInBunches Specifies whether or not the dispensed amount has to be presented in multiple bunches of items. Following values are possible: • `false` - The dispensed amount can be presented in one operation. • `true` - At least one additional bunch has to be presented. • `unknown` - It is unknown whether additional bunches will remain after the first present operation.

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

Properties
bunches If *presentInBunches* is "true", specifies the number of bunches of items to be presented as a result of the current operation. If not applicable or the number of bunches is at least two, but the precise number is unknown, *bunches* will be omitted.
token The dispense token that authorizes the dispense operation, as created by the authorizing host. See the generic end to end document for more information.

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 The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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 application 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
- CashDispenser.DelayedDispenseEvent
- CashDispenser.StartDispenseEvent
- CashManagement.CashUnitErrorEvent

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

- CashDispenser.ItemsTakenEvent
- CashDispenser.PartialDispenseEvent
- CashDispenser.SubDispenseOkEvent
- CashDispenser.IncompleteDispenseEvent
- CashManagement.NoteErrorEvent
- CashManagement.InfoAvailableEvent
- CashDispenser.ShutterStatusChangedEvent

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

6.2.6 CashDispenser.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 CashDispenser.Dispense command which caused these items to be dispensed will be used.

When this command successfully completes the items are in customer access.

If CashDispenser.Present was called as part of a sub-dispense operation, or if the previous Dispenser.Dispense command specified that the amount has to be presented in multiple bunches, the completion message includes details about remaining bunches. The field *additionalBunches* specifies, if there are any additional bunches to be dispensed to the customer and the field *bunchesRemaining* specifies the number of outstanding subdispense operations.

If the dispense operation is protected by end to end security then this command will cause the command nonce stored in the hardware to be cleared, which has the effect of invalidating any existing E2E tokens. Any future dispense operation will need to get a new command nonce and create a new dispense token.

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 canceled.
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	

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

Payload	Type	Required
`"position": "default",`	string	
`"additionalBunches": "none",`	string	
`"bunchesRemaining": 0`	integer	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

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

- CashDispenser.ItemsTakenEvent
- CashManagement.InfoAvailableEvent
- CashDispenser.ShutterStatusChangedEvent

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

6.2.7 CashDispenser.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.

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 canceled.
default: 0

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "cashUnitError"`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

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

6.2.8 CashDispenser.Retract

This command will retract items which may have been in customer access from an output position or from internal areas within the CashDispenser. 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 CashDispenser.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 canceled.

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.

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

Properties
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 The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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 CashDispenser.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
- CashDispenser.ItemsTakenEvent
- CashManagement.InfoAvailableEvent
- CashDispenser.IncompleteRetractEvent
- CashDispenser.ShutterStatusChangedEvent

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

6.2.9 CashDispenser.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 timeout but can be canceled.
default: 0

position
The output position where the shutter is to be opened. If the application 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",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "unsupportedPosition"`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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

- CashDispenser.ShutterStatusChangedEvent

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

6.2.10 CashDispenser.CloseShutter

This command closes 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 timeout but can be canceled.
default: 0

position
The output position where the shutter is to be closed. If the application 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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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

- CashDispenser.ShutterStatusChangedEvent

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

6.2.11 CashDispenser.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 applications 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 canceled.
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*.

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

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "invalidMixNumber"`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

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

6.2.12 CashDispenser.Reset

This command is used by the application 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 application 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.

If items are found inside the device the CashDispenser.MediaDetectedEvent will be generated and will inform the application 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 CashDispenser.Reset command was issued or the number counted by the device during the CashDispenser.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 canceled.

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.

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

Properties
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	
`}`		

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

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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 CashDispenser.IncompleteRetractEvent.

Event Messages

- CashManagement.CashUnitThresholdEvent
- CashManagement.CashUnitErrorEvent
- CashDispenser.MediaDetectedEvent
- CashDispenser.ItemsTakenEvent
- CashManagement.InfoAvailableEvent
- CashDispenser.IncompleteRetractEvent
- CashDispenser.ShutterStatusChangedEvent

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

6.2.13 CashDispenser.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 application 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": 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 canceled.

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.

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

Properties
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	

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

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)	

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

Payload	Type	Required
`},`		
`}`		
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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.

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

Properties
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 application 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 application 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 application 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.

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

Properties
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 canceled 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.

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

Properties
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.

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

Properties
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
- CashDispenser.ItemsTakenEvent
- CashManagement.CashUnitInfoChangedEvent
- CashManagement.NoteErrorEvent
- CashDispenser.ShutterStatusChangedEvent
- CashManagement.InfoAvailableEvent

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

6.2.14 CashDispenser.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 CashDispenser.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 canceled.
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	

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

Payload	Type	Required
`"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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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/counted
The number of items that were counted during the emptying of the cash unit.

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

Properties
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
- CashDispenser.ItemsTakenEvent
- CashDispenser.ItemsPresentedEvent
- CashManagement.NoteErrorEvent
- CashManagement.InfoAvailableEvent
- CashDispenser.ShutterStatusChangedEvent

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

6.2.15 CashDispenser.PrepareDispense

On some hardware it can take a significant amount of time for the CashDispenser to get ready to dispense media. On this type of hardware the CashDispenser.PrepareDispense command can be used to improve transaction performance.

If this command is supported (see the *prepareDispense* capability) then applications can help to improve the time taken to dispense media by issuing this command as soon as the application 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 CashDispenser.Denominate and CashDispenser.Dispense commands, which will not stop the dispense preparation, any execute command on CashDispenser 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 canceled.
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
The completion code.

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

Properties
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

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

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",
  "cashUnit": {
  "status": "ok",
  "type": "billCassette",
  "currencyID": Add example to YAML,
  "value": 0,
  "logicalCount": 0,
  "maximum": 0,
  "appLock": false,
  "cashUnitName": Add example to YAML,
  "initialCount": 0,
  "dispensedCount": 0,
  "presentedCount": 0,
  "retractedCount": 0,
  "rejectCount": 0,
  "minimum": 0,
  "physicalPositionName": Add example to YAML,
  "unitID": Add example to YAML,
  "count": 0,
  "maximumCapacity": 0,
  "hardwareSensor": false,
  "itemType": {
  "all": false,
  "unfit": false,
  "individual": false,
  "level1": false,
  "level2": false,
  "level3": false,
  "itemProcessor": false,
  "unfitIndividual": false
  },
	
	string
	object
	string
	string
	string
	number
	integer
	integer
	boolean
	string
	integer
	integer
	integer
	integer
	integer
	integer
	string
	string
	integer
	integer
	boolean
	object
	boolean
	boolean
	boolean
	boolean
	boolean
	boolean
	boolean
	boolean

Payload	Type	Required
`"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.

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

Properties
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 application 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 application 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 application 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.

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

Properties
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 canceled 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.

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

Properties
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.

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

Properties
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.

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

6.3.2 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 application 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	
`},`		

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

Payload	Type	Required
`"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 application to assign a value to this field. This value is persistent.

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

Properties
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 application 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 application 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.

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

Properties
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 canceled 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.

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

Properties
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.

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

6.3.3 CashDispenser.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.

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

6.3.4 CashDispenser.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.

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

6.3.5 CashDispenser.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.

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

6.3.6 CashDispenser.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 CashDispenser.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 application 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.

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

6.3.7 CashDispenser.IncompleteDispenseEvent

This event is generated during CashDispenser.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. CashDispenser.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 application 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.

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

6.3.8 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.

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

6.3.9 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.

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

6.3.10 CashDispenser.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`.

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

Properties
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.

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

6.3.11 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*):

CashDispenser.CalibrateCashUnit

CashDispenser.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 an application 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	

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

Payload	Type	Required
`"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.

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

Properties
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 application 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 application 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 application 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.

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

Properties
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 canceled 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.

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

Properties
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.

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

Properties
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.

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

6.4 Unsolicited Messages

6.4.1 CashDispenser.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.

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

6.4.2 CashDispenser.ItemsPresentedEvent

This event specifies that items have been presented to the user during a count operation and need to be taken.

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

6.4.3 CashDispenser.MediaDetectedEvent

This service event is generated if media is detected during a reset command. The payload on the event informs the application 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 ommitted.

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.

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

Properties
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.

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

6.4.4 CashDispenser.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.

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

7. Cash Acceptor Interface

This chapter defines the Cash Acceptor interface functionality and messages.

7.1 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.

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

7.2 Command Messages

7.2.1 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 canceled.
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.

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

Properties
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

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

7.2.2 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 canceled. 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.`		
`  }`		
`}`		

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

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

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

7.2.3 CashAcceptor.GetPositionCapabilities

This command allows the application 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 canceled. 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	

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

Payload	Type	Required
`}]`		
`}`		

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 application 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.

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

Properties
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

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

7.2.4 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

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

7.2.5 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 canceled.
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.

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

Properties
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

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

7.2.6 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 canceled. 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	
`      }`		
`    },`		
`  }`		
`}`		

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

Properties
cashUnitCaps Object containing additional cash unit capapabilities. Cash Unit capabiity 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 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.

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

Event Messages

None

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

7.2.7 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": [{`	array (object)	
`    "cashunitSource": Add example to YAML`	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

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

7.2.8 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 application 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 canceled.
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.

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

Properties
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

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

7.2.9 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 canceled. 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	
`"count": 0`	integer	
`}]`		
`},`		
`"totalReturnedItems": {`	object	
`See returnedItems properties.`		
`},`		
`"remainingItems": {`	object	
`See returnedItems properties.`		
`}`		
`}`		

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

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.

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

Event Messages

None

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

7.2.10 CashAcceptor.CashInStart

Before initiating a cash-in operation, an application 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 an application 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.

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

Properties
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

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "invalidTellerId"`	string	
`}`		

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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

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

7.2.11 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 canceled 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 application 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 application 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

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

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* == mixedMdeia) 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 canceled.
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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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 application 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

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

7.2.12 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 ItemProcesoor 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 canceled.

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	

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

Payload	Type	Required
"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)	
},		
}		
}		

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

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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 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.

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

Properties
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 application 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 application 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 application 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.

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

Properties
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 canceled 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.

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

Properties
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.

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

Properties
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

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

7.2.13 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 application 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 application 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 canceled.
default: 0

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "cashUnitError",`	string	

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

Payload	Type	Required
"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	
}]		

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

Payload	Type	Required
`},`		
`"noteIDs": [0]`	array (integer)	
`},`		
`}`		
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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 application 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.

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

Properties
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 application 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 application 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 application 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.

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

Properties
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 canceled 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.

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

Properties
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.

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

Properties
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
- CashManagement.InfoAvailableEvent
- CashManagement.CashUnitThresholdEvent
- CashAcceptor.ShutterStatusChangedEvent

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

7.2.14 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.

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

Properties
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	

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

Payload	Type	Required
`"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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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.

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

Properties
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 application 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 application 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 application 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.

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

Properties
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 canceled 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.

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

Properties
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
- CashManagement.NoteErrorEvent
- CashAcceptor.ItemsTakenEvent
- CashManagement.InfoAvailableEvent
- CashManagement.CashUnitInfoChangedEvent
- CashAcceptor.ShutterStatusChangedEvent

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

7.2.15 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 application 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
The completion code.

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

Properties
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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

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

7.2.16 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 application 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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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

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

7.2.17 CashAcceptor.Reset

This command is used by the application 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 application 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 application 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 application 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 application 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	
`}`		

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

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.

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

Properties
outputPosition The output position to which items are to be moved. This field is only used if *number* is zero and *netractArea* 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 The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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
- CashAcceptor.MediaDetectedEvent
- CashAcceptor.ItemsTakenEvent
- CashManagement.InfoAvailableEvent
- CashAcceptor.ShutterStatusChangedEvent

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

7.2.18 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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

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

7.2.19 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 application 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 application 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 application 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 canceled.
default: 0

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "tooManyItems",`	string	

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

Payload	Type	Required
`"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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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.

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

Properties
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

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

7.2.20 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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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.
"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

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

7.2.21 CashAcceptor.ComparePG6Signature

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.ComparePG6Signature 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.

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

Properties
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 emtpy) 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)	

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

Payload	Type	Required
`"index": 0,`	integer	
`"confidenceLevel": 0,`	integer	
`"comparisonData": Add example to YAML`	string	
`}]`		
`}`		

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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 application 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 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

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

7.2.22 CashAcceptor.Replenish

This command replenishes items from a single cash unit to multiple cash units. Applications 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.

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

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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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.

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

Properties
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

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

7.2.23 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 emtpy 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.Capabilites command.

amountLimit/currencyID

Currency identifier in ISO 4217 format [Ref. 2]. This must not be three ASCII 0x20 characters.

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

Properties
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 The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
errorCode Specifies the error code if applicable. Following values are possible: "exchangeActive": The device is in the exchange state.

Event Messages

None

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

7.2.24 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": Add example to YAML,`	string	
`"errorCode": "invalidCashUnit"`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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

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

7.2.25 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.

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

Properties
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 The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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

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

7.2.26 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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

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

7.2.27 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	

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

Payload	Type	Required
`"errorCode": "unsupportedPosition"`	string	
`}`		

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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
- CashAcceptor.ShutterStatusChangedEvent

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

7.2.28 CashAcceptor.Deplete

This command removes items from multiple cash units to a single cash unit. Applications 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": false`	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.

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

Completion Message

Payload	Type	Required
`{`		
`"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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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.

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

Properties
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

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

7.2.29 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:

OpenShutter

CashAcceptor.PresentMedia

The application 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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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

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

7.3 Event Messages

7.3.1 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",
  "cashUnit": {
  "status": "ok",
  "type": "billCassette",
  "currencyID": Add example to YAML,
  "value": 0,
  "logicalCount": 0,
  "maximum": 0,
  "appLock": false,
  "cashUnitName": Add example to YAML,
  "initialCount": 0,
  "dispensedCount": 0,
  "presentedCount": 0,
  "retractedCount": 0,
  "rejectCount": 0,
  "minimum": 0,
  "physicalPositionName": Add example to YAML,
  "unitID": Add example to YAML,
  "count": 0,
  "maximumCapacity": 0,
  "hardwareSensor": false,
  "itemType": {
  "all": false,
  "unfit": false,
  "individual": false,
  "level1": false,
  "level2": false,
  "level3": false,
  "itemProcessor": false,
  "unfitIndividual": false
  },
	
	string
	object
	string
	string
	string
	number
	integer
	integer
	boolean
	string
	integer
	integer
	integer
	integer
	integer
	integer
	string
	string
	integer
	integer
	boolean
	object
	boolean
	boolean
	boolean
	boolean
	boolean
	boolean
	boolean
	boolean

Payload	Type	Required
`"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.

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

Properties
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 application 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 application 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 application 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.

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

Properties
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 canceled 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.

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

Properties
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.

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

Properties
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.

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

7.3.2 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.

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

7.3.3 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.

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

7.3.4 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.

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

7.3.5 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.

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

7.3.6 CashAcceptor.InsertItemsEvent

This event notifies the application when the device is ready for the user to insert items.

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

7.3.7 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 application 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	
`},`		

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

Payload	Type	Required
`"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 application to assign a value to this field. This value is persistent.

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

Properties
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 application 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 application 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.

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

Properties
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 canceled 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.

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

Properties
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.

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

7.3.8 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*):

CashDispenser.CalibrateCashUnit

CashDispenser.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 an application 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	

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

Payload	Type	Required
`"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.

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

Properties
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 application 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 application 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 application 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.

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

Properties
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 canceled 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.

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

Properties
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.

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

Properties
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.

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

7.3.9 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.

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

7.3.10 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.

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

7.4 Unsolicited Messages

7.4.1 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.

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

7.4.2 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.

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

7.4.3 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.

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

7.4.4 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.

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

Properties
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 *netractArea* 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.

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

7.4.5 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.

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

7.4.6 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 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.

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

8. Key Management Interface

This chapter defines the Key Management interface functionality and messages.

8.1 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 verificationImportant 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. Applications can check impact of key deletion using Keydetail.

8.2 General Information

8.2.1 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
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 a device 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 device's public key at production time which could be for instance, the device manufacturer
Notation of Cryptographic Items and Functions	

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

Definitions and Abbreviations	
SK_E	The private key belonging to entity E
PK_E	The public belonging to entity E
SK_{DEVICE}	The private key belonging to the device
PK_{DEVICE}	The public key belonging to the device
SK_{HOST}	The private key belonging to the Host
PK_{HOST}	The public key belonging to the Host
SK_{SI}	The private key belonging to Signature Issuer
PK_{SI}	The public key belonging to Signature Issuer
SK_{ROOT}	The root private key belonging to the Host
PK_{ROOT}	The root public key belonging to the Host
K_{NAME}	A symmetric key
$Cert_{HOST}$	A Certificate that contains the public verification of the host and is signed by a trusted Certificate Authority.
$Cert_{DEVICE}$	A Certificate that contains the device public verification or encipherment key, which is signed by a trusted Certificate Authority.
$Cert_{CA}$	The Certificate of a new Certificate Authority
R_{DEVICE}	Random Number of the device
I_{HOST}	Identifier of the Host
K_{KTK}	Key Transport Key
R_{HOST}	Random number of the Host
I_{DEVICE}	Identifier of the device
TP_{DEVICE}	Thumb Print of the device
Sign (SK_E)[D]	The signing of data block D, using the private key SK_E
Recover (PK_E)[S]	The recovery of the data block D from the signature S, using the private key PK_E
RSACrypt (PK_E)[D]	RSA Encryption of the data block D using the public key PK_E
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 = (K_L || K_R), equivalent to Des (K_L) [Des^{-1} (K_R) [Des (K_L) [D]]]
$Des3^{-1}$ (K)[D]	Triple DES decipherment of an 8 byte data block D using the 16 byte secret key K = (K_L || K_R), equivalent to Des^{-1} (K_L) [Des (K_R) [Des^{-1} (K_L) [D]]]
Rnd_E	A random number created by entity E
UI_E	Unique Identifier for entity E
(A || B)	Concatenation of A and B

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

8.2.2 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 application 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 device:

1. Data block.
2. Digital signature.
3. Host's public key.

To validate the signature, the device performs the following:

1. The device 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. The device 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. The device compares digest1 with digest2.

If digest1 matches digest2 exactly, the device has confirmed the following:

- Data was not tampered with in transit. Changing a single bit in the data sent from the Host to the Key Management Device would cause digest2 to be different from digest1. Every data block has a unique digest; therefore, an altered data block is detected by the device.
- 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 device 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 Host and Device, inform each other of their Public Keys. This information is then used to securely send the Master Key to the Device. 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 (K_M) on the device 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.

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

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 signature Sign (SK_{SI})[PK_{ROOT}]. The Host has generated another key pair PK_{HOST} and SK_{HOST} and signs the PK_{HOST} with the SK_{ROOT}.
4. (Optional) The Host obtains a list of the valid device Unique Identifiers. The Signature Issuer installs a Signature Sign(SK_{SI})[UI_{DEVICE}] for the Unique ID (UI_{DEVICE}) on the Device. The Signature Issuer uses SK_{SI} to do this.
5. The Signature Issuer installs its Public Key (PK_{SI}) on the Device. It also derives and installs the Signature Sign(SK_{SI})[PK_{DEVICE}] of the Device's Public Key (PK_{DEVICE}) on the Device. The Signature Issuer uses SK_{SI} to do this.
6. The Device additionally contains its own Public (PK_{DEVICE}) and Private Key (SK_{DEVICE}).

Step 1

The Device sends its Public Key to the Host in a secure structure:

The Device sends its 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 Device Public Key.

The command used to export the device'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 Device Unique Identifier. The Device 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 Device Unique ID. It can then check this against the list it received from the Signature Issuer.

The command used to export the Device Unique Identifier is:

- KeyManagement.ExportRsaIssuerSignedItem.

Step 3 (Enhanced Remote Key Loading only)

The Host sends its root public key to the Device:

The Host sends its Root Public Key (PK_{ROOT}) and associated Signature. The Device verifies the signature using PK_{SI} and stores the key.

The command used to import the Host root public key securely as described above is:

- KeyManagement.ImportKey.

Step 4

The Host sends its public key to the Device:

The Host sends its Public Key (PK_{HOST}) and associated Signature. The Device verifies the signature using PK_{SI} (or PK_{ROOT} 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:

- KeyManagement.ImportKey.

Step 5

The Device receives its Master Key from the Host:

The Host encrypts the Master Key (K_M) with PK_{DEVICE}. A signature for this is then created using SK_{HOST}. The Device will then validate the signature using PK_{HOST} and then obtain the master key by decrypting using SK_{DEVICE}.

The commands used to exchange master symmetric keys as described above are:

- KeyManagement.StartKeyExchange

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

- KeyManagement.ImportKey

Step 6 — Alternative including random number

The Host requests the Device to begin the DES key transfer process and generate a random number.

The Host encrypts the Master Key (K_M) with PK_{DEVICE}. A signature for the random number and encrypted key is then created using SK_{HOST}.

The Device will then validate the signature using PK_{HOST}, verify the random number and then obtain the master key by decrypting using SK_{DEVICE}.

The commands used to exchange master symmetric keys as described above are:

- KeyManagement.StartKeyExchage
- KeyManagement.ImportKey

Initialization Phase – Signature Issuer and ATM PIN

This would typically occur in a secure manufacturing environment.

Initialization Phase – Signature Issuer and Host

This would typically occur in a secure offline environment.

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

Key Exchange – Host and ATM PIN

This following is a typical interaction for the exchange of the initial symmetric master key between host and device.

The following is the recommended sequence of interchanges.

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

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 device supports the KeyManagement.StartKeyExchange command.

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

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

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 host and device supports the Enhanced Signature Remote Key Loading scheme.

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

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

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 applications 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 issuer, i.e. PK_{SI}
"_EPPCryptKey"	Public/Private key-pair	The private key associated with _SigIssuerVendor	The key-pair used to encrypt and encrypt the symmetric. key, i.e SK_{DEVICE} and PK_{DEVICE}. The public key is used for encryption by the host and the private for decryption by the EPP.
"_DeviceCryptCert"	Public/Private key-pair	CA	This key is used for certificate based remote key loading when transporting synmetric key. The private key is used for decryption by the device. i.e. $Cert_{DEVICE}$
"_HostCert"	Public Key	CA	The certificate issued by the host, which contains a public key to verify the certificate. i.e. $Cert_{HOST}$

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.

8.2.3 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.

Certificate Exchange and Authentication

In summary, both end points, the device and the Host, inform each other of their Public Keys. This information is then used to securely send the Master Key to the device. 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 commands do not necessarily need to be called in the order below. This way though is the recommended 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 device 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 device stores the HOST's Public Verification Key.

• Next, KeyManagement.GetCertificate is called. The device 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 device's verification or encryption key (primary or secondary this depends on the state of the device).

The following diagram shows how the Host and ATM Load and Get each other's information to make Remote Key Loading possible:

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

Remote Key Exchange

After the above has been completed, the host is ready to load the key into the device. The following is done to complete this and the application must complete the Remote Key Exchange in this order:

1. First, the Keymanagement.StartKeyExchange is called. This returns R_{DEVICE} from the device to be used in the authenticating the Keymanagement.ImportRSAEnchiperdPKCS7Key message.
2. Next, Keymanagement.ImportKey is called. This command sends down the KTK to the device. The following items below show how this is accomplished.a) Host has obtained a Key Transport Key and wants to transfer it to the device. Host constructs a key block containing an identifier of the host, I_{HOST}, and the key, K_{KTK}, and enciphers the block, using the device'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, R_{HOST}, the random number of the device returned in the Keymanagement.StartKeyExchange command, R_{DEVICE}, the identifier of the encryptor, I_{ENC}, and the enciphered key block. The host signs the whole block using its private signature key and sends the message down to the device. The device then verifies the host's signature on the message by using the host's Public Verification Key. Then the device checks the identifier and the random number of the device passed in the message to make sure that the device is talking to the right host. The device then deciphers the enciphered block using its private verification key. After the message has been deciphered, the device checks the Identifier of the host. Finally, if everything checks out to this point the device will load the Key Transport Key. NOTE: If one step of this verification occurs the device will return the proper error to the host.c) After the Key Transport Key has been accepted, the device constructs a message that contains the random number of the host, the random number of the device and the host identifier all signed by the private signature key of the device. This message is sent to the host.d)

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

The host verifies the message sent from the device by using the device'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 device's random number with the one received in the Keymanagement.StartKeyExchange command.

The following diagram below shows how the host and device transmit the Key Transport Key.

Replace Certificate

After the key has been loaded into the device, the following can be completed:

- (Optional) KeyManagement.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 device accepts the new Certificate Authority. The host sends the message to the device. The device uses the host's Public Verification Key to verify the host's signature. The device 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 Device communicate to load the new CA.

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

Primary and Secondary Certificate

Primary and Secondary Certificates for both the Public Verification Key and Public Encipherment Key are pre-loaded into the device. 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 KeyManagement.LoadCertificate or 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 device to shift to the secondary certificate state, only Secondary Certificates can be used. The device 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 device should be handled with the manufacturer.

8.2.4 Remote Key Loading Using TR34

TR34 BIND To Host

This section defines the commands to use when transferring a TR34 BIND token.

This step is a pre-requisite for all other TR34 operations. The device must be bound to a host before any other TR34 operation will succeed.

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

NB: While the device encryption certificate is not required to build the BIND token, it is recommended that the encryption certificate is retrieved during this process and is stored for future use. Otherwise, if not stored, it will need to be requested prior to all other TR34 token transfer requests.

TR34 Key Transport

There are two protocols 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: The crklLoadOptions capability indicates which protocol the device supports.

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 device 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 device is bound to the host.

NB: Dotted lines represent commands that are only required if the PIN device encryption certificate has not been previously stored by the host.

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.

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

NB: Dotted lines represent commands that are only required if the device encryption certificate has not been previously stored by the host.

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 device is bound to the host.

NB: Dotted lines represent commands that are only required if the device 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.

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 device is bound to the host.

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

NB: Dotted lines represent commands that are only required if the device encryption certificate has not been previously stored by the host.

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 device is bound to the host.

NB: Dotted lines represent commands that are only required if the device 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.

8.2.5 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.

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

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 application is responsible for management of the key lifetime and expiry after the key is successfully imported

Keys Loading

The final goal of an application 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 application should first load the keys issued by the Certification Authority. At transaction time the application will use these keys to load the keys that the application has retrieved from the chip card.

Certification Authority keys

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).

EPI CA format

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	Binar
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 Remainder	37	37 least significant bytes of the Europay public key Modulus	Binary
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.

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

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.

Chip card keys

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 application is responsible to send the PIN block to the chip card inside the right APDU.

SHA-1 Digest

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 application will pass the data to be hashed to the Service Provider. Once the device completes the SHA-1 hash code, the Service Provider will return the 20-byte hash value back to the application.

8.2.6 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 = encrypted key value>
Signature = signature generated by the host>
Usage of the key to be imported = key encrypting key
RSA Encipher Algorithm = RSAES-OAEP
RSA Signature Algorithm = RSASSA-PSS

KeyManagement.ImportKey command payload

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

```
{
  "header": {
    "type": "command",
    "name": "KeyManagement.ImportKey",
    "requestId": 12345
  },
  "payload" : {
    "key": "testKey","
    "keyAttributes": {
      "keyUsage": "K0",
      "algorithm": "T",
      "modeOfUse": "D",
    },
    "value": "<<encrypted key value>",
    "decryptKey": "eppCryptKey",
    "decryptKey": "rsaesOaep",
    "verificationData": "<signature generated by the host>",
    "verifyKey": "hostKey",
    "verifyAttributes": {
      "cryptoMethod": "rsassaPss",
      "hashAlgorithm": "sha256"
    },
  }
}
```

For this example, the following output data is expected:

Key Check Mode = kcv zero
Key Check Value = key check value>
Key Length = double length key

KeyManagement.ImportKey completion payload

```
{
  "header": {
    "type": "completion",
    "name": "KeyManagement.ImportKey",
    "requestId": 12345
  },
  "payload" : {
    "verificationData": "<key check value>",
    "verifyAttributes": {
      "keyUsage": "00",
      "algorithm": "T",
      "modeOfUse": "V",
      "cryptoMethod": "kcvZero"
    },
    "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
Name of the key used to verify the encrypted key value = verifyKey and key usage is '00'
Encrypted key value = encrypted key value>
Usage of the key to be imported = pin encryption
Key Check Mode = KCV Zero
Key Check Value = key check value>

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

KeyManagement.ImportKey command payload

```
{
  "header": {
    "type": "command",
    "name": "KeyManagement.ImportKey",
    "requestId": 12345
  },
  "payload" : {
    "key": "testKey","
    "keyAttributes": {
      "keyUsage": "P0",
      "algorithm": "T",
      "modeOfUse": "E",
    },
    "value": "<encrypted key value>",
    "decryptKey": "masterKey",
    "decryptMethod": "ecb",
    "verificationData": "<key check value encoded>",
    "verifyKey": "verifyKey",
    "verifyAttributes": {
      "cryptoMethod": "kcvZero"
    }
  }
}
```

Likewise, the following output data is expected:

KeyManagement.ImportKey completion payload

```
{
  "header": {
    "type": "completion",
    "name": "KeyManagement.ImportKey",
    "requestId": 12345
  },
  "payload" : {
    "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 = encrypted key value>
Usage of the key to be imported = MAC

KeyManagement.ImportKey command payload

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

```
{
  "header": {
    "type": "command",
    "name": "KeyManagement.ImportKey",
    "requestId": 12345
  },
  "payload" : {
    "key": "testKey","
    "keyAttributes": {
      "keyUsage": "M3",
      "algorithm": "T",
      "modeOfUse": "G",
  },
  "value": "<encrypted key value encoded>",
  "decryptKey": "masterKey",
  "decryptMethod": "ecb"
}
```

KeyManagement.ImportKey completion payload

```
{
  "header": {
    "type": "completion",
    "name": "KeyManagement.ImportKey",
    "requestId": 12345
  },
  "payload" : {
    "verificationData": "<key check value>"
    "verifyAttributes": {
      "keyUsage": "00",
      "algorithm": "T",
      "modeOfUse": "V",
      "cryptoMethod": "kcvZero"
    },
    "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 = key value>
Signature = signature generated by the vendor signature issuer>
Usage of the key to be imported = RSA signature verification
RSA Signature Algorithm = RSA SSA PSS

KeyManagement.ImportKey command payload

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

```
{
  "header": {
    "type": "command",
    "name": "KeyManagement.ImportKey",
    "requestId": 12345
  },
  "payload" : {
    "key": "hostKey","
    "keyAttributes": {
      "keyUsage": "S0",
      "algorithm": "R",
      "modeOfUse": "V",
    },
    "value": "<key value>",
    "verificationData": "<signature generated by the vendor signature issuer>",
    "verifyKey": "sigIssuerVendor",
    "verifyAttributes": {
      "cryptoMethod": "rsassaPss",
      "hashAlgorithm": "sha256"
    }
  }
}
```

For this example, the following output data is expected:

RSA Key Check Mode = sha256 digest
Key Check Value = sha256 digest>
Key Length = 2048

KeyManagement.ImportKey completion payload

```
{
  "header": {
    "type": "completion",
    "name": "KeyManagement.ImportKey",
    "requestId": 12345
  },
  "payload" : {
    "verificationData": "<sha256 digest>",
    "verifyAttributes": {
      "keyUsage": "S0"
      "algorithm": "R",
      "modeOfUse": "V",
      "hashAlgorithm": "sha256"
    },
    "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 = key block>

KeyManagement.ImportKey command payload

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

```
{
  "header": {
    "type": "command",
    "name": "KeyManagement.ImportKey",
    "requestId": 12345
  },
  "payload" : {
    "key": "testKey","
    "keyAttributes": {
      "keyUsage": "D0",
      "algorithm": "T",
      "modeOfUse": "E",
    },
    "value": "<key block>",
    "decryptKey": "masterKey"
  }
}
```

For this example, the following output data is expected:

Key Length = triple length (192 bits) DES key

KeyManagement.ImportKey completion payload

```
{
  "header": {
    "type": "completion",
    "name": "KeyManagement.ImportKey",
    "requestId": 12345
  },
  "payload" : {
    "keyLength": 192
  }
}
```

8.2.7 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: Application cryptograms	rsaPublicVerify
“E1”	“X”	EMV/chip Issuer Master Key: Secure Messaging for Confidentiality	rsaPublicVerify

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

TR-31 Value	TR-31 Mode(s) of use	Definition	use
“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.

8.2.8 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:

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

New master keys loaded with restrictedKeyEncKey flag, encrypted with themselves

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

Host → KeyManagement

ImportKey
[crypt master key values are loaded encrypted with previous crypt master key values]

```
{
  "header": {
    "type": "command",
    "name": "KeyManagement.ImportKey",
    "requestId": 12351
  },
  "payload" : {
    "key": "MasterKey1",
    "keyAttributes": {
      "keyUsage": "K0",
      "restricted": "D0",
      "algorithm": "T",
      "modeOfUse": "B"
    },
    "value": "<key data>",
    "decryptKey": "MasterKey1",
    "decryptMethod": "ecb"
  }
}
```

ImportKey
[function master key values are loaded encrypted with previous function master key values]

```
{
  "header": {
    "type": "command",
    "name": "KeyManagement.ImportKey",
    "requestId": 12352
  },
  "payload" : {
    "key": "MasterKey2",
    "keyAttributes": {
      "keyUsage": "K0",
      "restricted": "P0",
      "algorithm": "T",
      "modeOfUse": "E"
    },
    "value": "<key data>",
    "decryptKey": "MasterKey2",
    "decryptMethod": "ecb"
  }
}
```

Host → KeyManagement

Loading derived keys:

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

Host → KeyManagement

ImportKey
[crypt encryption key is derived from the crypt master encryption key]

```
{
  "header": {
    "type": "command",
    "name": "KeyManagement.ImportKey",
    "requestId": 12353
  },
  "payload" : {
    "key": "CryptKey",
    "keyAttributes": {
      "keyUsage": "D0",
      "algorithm": "T",
      "modeOfUse": "B"
    },
    "value": "<key data>",
    "decryptKey": "MasterKey1",
    "decryptMethod": "ecb"
  }
}
```

ImportKey
[function encryption key is derived from the function the master encryption key]

```
{
  "header": {
    "type": "command",
    "name": "KeyManagement.ImportKey",
    "requestId": 12354
  },
  "payload" : {
    "key": "FunctionKey",
    "keyAttributes": {
      "keyUsage": "P0",
      "algorithm": "T",
      "modeOfUse": "E"
    },
    "value": "<key data>",
    "decryptKey": "MasterKey2",
    "decryptMethod": "ecb"
  }
}
```

Host → KeyManagement

Usage sample for derived keys

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

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

Master key restriction disallows loading of derived keys with different usage:

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

8.3 Command Messages

8.3.1 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 applications. 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 canceled. 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	
`"keyDetails": {`	object	
`"exampleProperty1": {`	object	
`"generation": 0,`	integer	
`"version": 0,`	integer	
`"activatingDate": Add example to YAML,`	string	
`"expiryDate": Add example to YAML,`	string	
`"loaded": "no",`	string	Yes
`"keyBlockInfo": {`	object	
`"keyUsage": Add example to YAML,`	string	Yes
`"restrictedKeyUsage": Add example to YAML,`	string	
`"algorithm": Add example to YAML,`	string	Yes

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

Payload	Type	Required
`"modeOfUse": Add example to YAML,`	string	Yes
`"keyVersionNumber": Add example to YAML,`	string	
`"exportability": Add example to YAML,`	string	Yes
`"optionalBlockHeader": Add example to YAML,`	string	
`"keyLength": 0`	integer	Yes
`}`		
`},`		
`"exampleProperty2": {`	object	
`See exampleProperty1 properties.`		
`}`		
`}`		
`}`		

Properties

completionCode

The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription

If included, this contains additional vendor dependent information to assist with problem resolution.

errorCode

Specifies the error code if applicable. The following values are possible:

- `keyNotFound` - The specified key name is not found.

keyDetails

This property contains key/value pairs where the key is a name of key and the value is the key detail.

keyDetails/exampleProperty1 (example name)

The object contains key detail.

keyDetails/exampleProperty1/generation

Specifies the generation of the key. Different generations might correspond to different environments (e.g. test or production environment). The content is vendor specific. This value can be omitted if no such information is available for the key.

Property value constraints:

```
minimum: 0
maximum: 99
```

keyDetails/exampleProperty1/version

Specifies the version of the key (the year in which the key is valid, e.g. 1 for 2001). This value can be omitted if no such information is available for the key.

Property value constraints:

```
minimum: 0
maximum: 99
```

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

Properties
keyDetails/exampleProperty1/activatingDate Specifies the date when the key is activated in the format YYYYMMDD. This value can be omitted if no such information is available for the key. Property value constraints: `pattern: ^[0-9]{4}(0[1-9]|1[0-2])(0[1-9]|[12][0-9]|3[01])$`
keyDetails/exampleProperty1/expiryDate Specifies the date when the key expires in the format YYYYMMDD. This value can be omitted if no such information is available for the key. Property value constraints: `pattern: ^[0-9]{4}(0[1-9]|1[0-2])(0[1-9]|[12][0-9]|3[01])$`
keyDetails/exampleProperty1/loaded Specifies whether the key has been loaded (imported from Application or locally from Operator). • `no` - The key is not loaded. • `yes` - The key is loaded and ready to be used in cryptographic operations. • `unknown` - The State of the key is unknown. • `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.
keyDetails/exampleProperty1/keyBlockInfo Specifies the key attributes using ANSI TR-31 keyblock header definitions.

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

Properties

keyDetails/exampleProperty1/keyBlockInfo/keyUsage

Specifies the intended function of the key. See [Reference 35. ANS X9 TR-31 2018] for all possible values. 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` - EMV / Chip Issuer Master Key: Application 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 5.
- `M6` - ISO 9797-1:2011 MAC Algorithm 5 / CMAC.
- `M7` - HMAC.
- `M8` - ISO 9797-1:2011 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.
- `00 - 99` - These numeric values are reserved for proprietary use.

Property value constraints:

```
pattern: ^B[0-2]$|^C0$|^D[0-2]$|^E[0-6]$|^I0$|^K[0-3]$|^M[0-8]$|^P0$|^S[0-2]$|^V[0-4]$|^[0-9][0-9]$
```

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

Properties

keyDetails/exampleProperty1/keyBlockInfo/restrictedKeyUsage

Specifies restricted key usage of the key associated with the keyUsage property specified. This property can be omitted if there is no restricted key usage required. Following restricted key usage can be set if the keyUsage property specifies either K0 or K1. 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` - EMV / Chip Issuer Master Key: Application 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).
- `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 5.
- `M6` - ISO 9797-1:2011 MAC Algorithm 5 / CMAC.
- `M7` - HMAC.
- `M8` - ISO 9797-1:2011 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.
- `00 – 99` - These numeric values are reserved for proprietary use.

Property value constraints:

```
pattern: ^B[0-2]$|^C0$|^D[0-2]$|^E[0-6]$|^I0$|^K[2-3]$|^M[0-8]$|^P0$|^S[0-2]$|^V[0-4]$|^[0-9][0-9]$
```

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

Properties
keyDetails/exampleProperty1/keyBlockInfo/algorithm Specifies the algorithm for which the key can be used. See [Reference 35. ANS X9 TR-31 2018] for all possible values. The following values are possible: • `A` - AES. • `D` - DEA. • `E` - Elliptic Curve. • `H` - HMAC. • `R` - RSA. • `S` - DSA. • `T` - Triple DEA (also referred to as TDEA). • `0 - 9` - These numeric values are reserved for proprietary use. Property value constraints: `pattern: ^[0-9ADEHRST]$`
keyDetails/exampleProperty1/keyBlockInfo/modeOfUse Specifies the operation that the key can perform. See [Reference 35. ANS X9 TR-31 2018] for all possible values. The following values are possible: • `B` - Both Encrypt and Decrypt / Wrap and unwrap. • `C` - Both Generate and Verify. • `D` - Decrypt / Unwrap Only. • `E` - Encrypt / Wrap Only. • `G` - Generate Only. • `N` - No special restrictions. • `S` - Signature Only. • `T` - Both Sign and Decrypt. • `V` - Verify Only. • `X` - Key used to derive other keys(s). • `Y` - Key used to create key variants. • `0 - 9` - These numeric values are reserved for proprietary use. Property value constraints: `pattern: ^[0-9BCDEGNSTVXY]$`
keyDetails/exampleProperty1/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. This value can be omitted if Key versioning is not used. Property value constraints: `pattern: ^[0-9a-zA-Z][0-9a-zA-Z]$`
keyDetails/exampleProperty1/keyBlockInfo/exportability Specifies whether the key may be transferred outside of the cryptographic domain in which the key is found. See [Reference 35. ANS X9 TR-31 2018] for all possible values. The following values are possible: • `E` - Exportable under a KEK in a form meeting the requirements of X9.24 Parts 1 or 2. • `N` - Non-exportable by the receiver of the key block, or from storage. Does not preclude exporting keys derived from a non-exportable key. • `S` - Sensitive, Exportable under a KEK in a form not necessarily meeting the requirements of X9.24 Parts 1 or 2. • `0 - 9` - These numeric values are reserved for proprietary use. Property value constraints: `pattern: ^[0-9ESN]$`

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

Properties
keyDetails/exampleProperty1/keyBlockInfo/optionalBlockHeader Contains any optional header blocks, as defined in [Reference 35. ANS X9 TR-31 2018]. This value can be omitted if there are no optional block headers.
keyDetails/exampleProperty1/keyBlockInfo/keyLength Specifies the length, in bits, of the key. 0 if the key length is unknown.

Event Messages

None

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

8.3.2 KeyManagement.Initialization

The encryption module must be initialized before any encryption function can be used. Every call to KeyManagement.Initialization destroys all application 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 application 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' application 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 application 'initial' keys would normally get updated by the application 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 application 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 application must await the KeyManagement.InitializedEvent.

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	
`"authentication": {`	object	
`"method": "none",`	string	
`"key": Add example to YAML,`	string	
`"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 canceled.

default: 0

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

Properties
authentication This property can be used to add authentication to perform an authenticated action. The KeyManagement.StartAuthenticate command must be called before this property is used by the command required authenticated action. If this property set without first calling the KeyManagement.StartAuthenticate command, then this command will fail and sequenceError will be returned. The KeyManagement.StartAuthenticate command does not need to immediately precede the command with this property. It is acceptable for other commands to be executed until this property is set since KeyManagement.StartAuthenticate command is called, except for any command that will clear from the KeyManagement interface the data that is being saved in order to verify the signed data provided in the command with this property. If this occurs, then sequenceError will be returned.
authentication/method Specifies the signer of the data, with one of the following values: • `none` - Authentication is not required. • `certhost` - The data is signed by the current Host, using the RSA certificate-based scheme. • `sigHost` - The data is signed by the current Host, using the RSA signature-based scheme. • `ca` - The data is signed by the Certificate Authority (CA). • `hl` - The data is signed by the Higher Level (HL) Authority. • `cbcmac` - A MAC is calculated over the data using *key* property and the CBC MAC algorithm. • `cmac` - A MAC is calculated over the data using *key* and the CMAC algorithm. • `reserved1` - Reserved for a vendor-defined signing method. • `reserved2` - Reserved for a vendor-defined signing method. • `reserved3` - Reserved for a vendor-defined signing method.
authentication/key If the *signer* is cbcmac or mac are specified, then this *signatureKey* property is the name of a key with the key usage of key attribute is M0 to M8. If sigHost is specified for signer property, then this property signatureKey specifies the name of a previously loaded asymmetric key(i.e. an RSA Public Key). The default Signature Issuer public key (installed in a secure environment during manufacture) will be used, if this signatureKey propery is omitted or contains the name of the default Signature Issuer as defined in the document Default keys and securitry item loaded during manufacture. Otherwise, this property should be omitted.
authentication/data This property contains the signed version of the base64 encoded data that was provided by the KeyManagement device during the previous call to the StartExchange command. The signer specified by *signer* property is used to do the signing. Both the signature and the data that was signed must be verified before the operation is performed. If certHost, ca, or hl are specified for *signer* property, then *signedData* is a PKCS #7 structure which includes the data that was returned by the StartAuthenticate command. The optional CRL field may or may not be included in the PKCS #7 *signedData* structure. If the signer is certHostTr34, caTr34 or hlTr34, please refer to the X9 TR34-2012 [Ref. 42] for more details. If sigHost is specified for the *signer* property specified, then s is a PKCS #7 structure which includes the data that was returned by the StartAuthenticate command. If cmcmac or cmac are specified for the *signer* property specified, then *signatureKey* must refer to a key loaded with the key usage of key attribute is M0 to M8.

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "accessDenied"`	string	

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

Payload	Type	Required
`}`		

Properties
completionCode The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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.

Event Messages

None

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

8.3.3 KeyManagement.DeriveKey

A key is derived from input data using a key generating key and an initialization vector. The input data can be expanded with a fill-character to the necessary length (mandated by the encryption algorithm being used). The derived key is imported into the encryption module and can then be used for further operations.

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	
`}`		

Properties

timeout
Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be canceled.
default: 0

derivationAlgorithm
Specifies the algorithm that is used for derivation. Possible values are: (see derivationAlgorithms)

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 omitted, *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 range is 0 to 255.
Property value constraints:

```
minimum: 0
maximum: 255
```

inputData
Data to be used for key derivation.

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

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "keyNotFound"`	string	
`}`		

Properties

completionCode

The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription

If included, this contains additional vendor dependent information to assist with problem resolution.

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

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

8.3.4 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 canceled.
default: 0

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

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

8.3.5 KeyManagement.ImportKey

The encryption key passed by the application 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	
`"restricted": Add example to YAML`	string	
`},`		
`"value": Add example to YAML,`	string	
`"constructing": false,`	boolean	
`"decryptKey": Add example to YAML,`	string	
`"decryptMethod": "ecb",`	string	
`"verificationData": Add example to YAML,`	string	
`"verifyKey": Add example to YAML,`	string	
`"verifyAttributes": {`	object	
`"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 canceled.
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 keyAttribute capability. The values specified must be compatible with the key identified by key.

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

Properties

keyAttributes/keyUsage

Specifies the key usage. 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` - EMV / Chip Issuer Master Key: Application 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 5.
- `M6` - ISO 9797-1:2011 MAC Algorithm 5 / CMAC.
- `M7` - HMAC.
- `M8` - ISO 9797-1:2011 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.
- `00 – 99` - These numeric values are reserved for proprietary use.

Property value constraints:

```
pattern: ^B[0-2]$|^C0$|^D[0-2]$|^E[0-6]$|^I0$|^K[0-3]$|^M[0-8]$|^P0$|^S[0-2]$|^V[0-
4]$|^[0-9][0-9]$
```

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

Properties
keyAttributes/algorithm Specifies the encryption algorithm. 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. Property value constraints: `pattern: ^[0-9ADRT]$`
keyAttributes/modeOfUse Specifies the encryption mode. The following values are possible: • `B` - Both Encrypt and Decrypt / Wrap and unwrap. • `C` - Both Generate and Verify. • `D` - Decrypt / Unwrap Only. • `E` - Encrypt / Wrap Only. • `G` - Generate Only. • `S` - Signature Only. • `T` - Both Sign and Decrypt. • `V` - Verify Only. • `X` - Key used to derive other keys(s). • `Y` - Key used to create key variants. • `0 – 9` - These numeric values are reserved for proprietary use. Property value constraints: `pattern: ^[0-9BCDEGSTVXY]$`

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

Properties

keyAttributes/restricted

This property should only be included if the keyUsage is an exchange key (K0, K1 or K2) and the key can only be used as the decryptKey for keys with one of the following usages:

- `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` - EMV / Chip Issuer Master Key: Application 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 5.
- `M6` - ISO 9797-1:2011 MAC Algorithm 5 / CMAC.
- `M7` - HMAC.
- `M8` - ISO 9797-1:2011 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.
- `00 - 99` - These numeric values are reserved for proprietary use.

Property value constraints:

```
pattern: ^B[0-2]$|^C0$|^D[0-2]$|^E[0-6]$|^I0$|^K[0-3]$|^M[0-8]$|^P0$|^S[0-2]$|^V[0-4]$|^[0-9][0-9]$
```

value

Specifies the Base64 encoded value of key to be loaded. If it is an RSA key the first 4 bytes contain the exponent and the following 128 the modulus. This property is not required for secure key entry and can be omitted.

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

Properties
constructing If the key is under construction through the import of multiple parts from a secure encryption key entry buffer, then this property is set to true. default: false
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 or the constructing property is true.
decryptMethod Specifies the cryptographic method that shall be used with the key specified by *decryptKey*. This property is not required if a keyblock is being imported, as the decrypt method is contained within the keyblock. This property specifies the cryptographic method that will be used to decrypt the encrypted value. This property is not required if the *constructing* property is true or if *decryptKey* is omitted. For a list of valid values see this property in the decryptAttribute capability. If the decryptKey algorithm is 'A', 'D', or 'T', then this property 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 the decryptKey algorithm is 'R', then this property can be one of the following values: • `rsaesPkcs1V15` - Use the RSAES_PKCS1-v1.5 algorithm. • `rsaesOaep` - Use the RSAES OAEP algorithm. If the specified decryptKey is key usage 'K1', then this property can be omitted. TR-31 defines the cryptographic methods used for each key block version.
verificationData Contains the data to be verified before importing. This property can be omitted if no verification is needed before importing the key, the *constructing* property is true or *value* contains verification data.
verifyKey Specifies the name of the previously loaded key which will be used to verify the *verificationData*. This property can be omitted when no verification is needed before importing the key or the *constructing* property is true.
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 verifyAttributes capability. This property can be omitted if *verificationData* is not required or the *constructing* property is true.

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

Properties
verifyAttributes/cryptoMethod This parameter specifies the cryptographic method cryptomethod that will be used with encryption algorithm. If the verifyKey algorithm is 'A', 'D', or 'T' and specified verifyKey is MAC key usage (i.e. 'M1'), then this property can be omitted. If the verifyKey algorithm is 'A', 'D', or 'T' and specified verifyKey is key usage '00', then this property 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 the verifyKey algorithm is 'R' and specified verifyKey is not key usage '00', then this property can be one of the following values: • `sigNone` - No signature algorithm specified. No signature verification will take place and the content of verificationData is not required. • `rsassaPkcs1V15` - Use the RSASSA-PKCS1-v1.5 algorithm. • `rsassaPss` - Use the RSASSA-PSS algorithm.
verifyAttributes/hashAlgorithm For asymmetric signature verification methods (Specified verifyKey usage is 'S0', 'S1', or 'S2'), this can be one of the following values to be used. If the specified verifyKey is key usage any of the MAC usages (i.e. 'M1'), then this property can be omitted. • `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.

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": "sha1"`	string	
`},`		
`"keyLength": 0`	integer	
`}`		

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

Properties
completionCode The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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 can be omitted 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 verifyAttributes capability fields. This field is not set if there is no verification data.

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

Properties
verifyAttributes/keyUsage Specifies the key usage. 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` - ISO 9797-1:2011 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. Property value constraints: `pattern: ^M[0-8]$\|^S[0-2]$|^[0-9][0-9]$`
verifyAttributes/algorithm Specifies the encryption algorithm. The following values are possible: • `A` - AES. • `D` - DEA. • `R` - RSA. • `T` - Triple DEA (also referred to as TDEA). • `"0" – "9"` - These numeric values are reserved for proprietary use. Property value constraints: `pattern: ^[0-9ADRT]$`
verifyAttributes/modeOfUse Specifies the encryption mode. The following values are possible: • `S` - Signature. • `V` - Verify Only. • `0 – 9` - These numeric values are reserved for proprietary use. Property value constraints: `pattern: ^[0-9SV]$`

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

Properties
verifyAttributes/cryptoMethod This parameter specifies the cryptographic method cryptomethod that will be used with encryption algorithm. If the *algorithm* property is 'A', 'D', or 'T' and specified *keyUsage* property is MAC key usage (i.e. 'M1'), then this property can be omitted. If the *algorithm* property is 'A', 'D', or 'T' and specified *keyUsage* property is '00', then this property 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 the *algorithm* property is 'R' and specified *keyUsage* property is not '00', then this property can be one of the following values: • `sigNone` - No signature algorithm specified. No signature verification will take place and the content of verificationData is not required. • `rsassaPkcs1V15` - Use the RSASSA-PKCS1-v1.5 algorithm. • `rsassaPss` - Use the RSASSA-PSS algorithm.
verifyAttributes/hashAlgorithm For asymmetric signature verification methods (Specified *keyUsage* property is 'S0', 'S1', or 'S2'), this can be one of the following values to be used. If the *keyUsage* property is any of the MAC usages (i.e. 'M1'), then this property can be omitted. • `sha1` - The SHA 1 digest algorithm. • `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

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

8.3.6 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.

Command Message

Payload	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"authentication": {`	object	
`"method": "none",`	string	
`"key": Add example to YAML,`	string	
`"data": 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 canceled.

default: 0

authentication

This property can be used to add authentication to perform an authenticated action. The KeyManagement.StartAuthenticate command must be called before this property is used by the command required authenticated action. If this property set without first calling the KeyManagement.StartAuthenticate command, then this command will fail and sequenceError will be returned. The KeyManagement.StartAuthenticate command does not need to immediately precede the command with this property. It is acceptable for other commands to be executed until this property is set since KeyManagement.StartAuthenticate command is called, except for any command that will clear from the KeyManagement interface the data that is being saved in order to verify the signed data provided in the command with this property. If this occurs, then sequenceError will be returned.

authentication/method

Specifies the signer of the data, with one of the following values:

- `none` - Authentication is not required.
- `certhost` - The data is signed by the current Host, using the RSA certificate-based scheme.
- `sigHost` - The data is signed by the current Host, using the RSA signature-based scheme.
- `ca` - The data is signed by the Certificate Authority (CA).
- `hl` - The data is signed by the Higher Level (HL) Authority.
- `cbcmac` - A MAC is calculated over the data using *key* property and the CBC MAC algorithm.
- `cmac` - A MAC is calculated over the data using *key* and the CMAC algorithm.
- `reserved1` - Reserved for a vendor-defined signing method.
- `reserved2` - Reserved for a vendor-defined signing method.
- `reserved3` - Reserved for a vendor-defined signing method.

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

Properties
authentication/key If the *signer* is cbcmac or mac are specified, then this *signatureKey* property is the name of a key with the key usage of key attribute is M0 to M8. If sigHost is specified for signer property, then this property signatureKey specifies the name of a previously loaded asymmetric key(i.e. an RSA Public Key). The default Signature Issuer public key (installed in a secure environment during manufacture) will be used, if this signatureKey propery is omitted or contains the name of the default Signature Issuer as defined in the document Default keys and securitry item loaded during manufacture. Otherwise, this property should be omitted.
authentication/data This property contains the signed version of the base64 encoded data that was provided by the KeyManagement device during the previous call to the StartExchange command. The signer specified by *signer* property is used to do the signing. Both the signature and the data that was signed must be verified before the operation is performed. If certHost, ca, or hl are specified for *signer* property, then *signedData* is a PKCS #7 structure which includes the data that was returned by the StartAuthenticate command. The optional CRL field may or may not be included in the PKCS #7 *signedData* structure. If the signer is certHostTr34, caTr34 or hlTr34, please refer to the X9 TR34-2012 [Ref. 42] for more details. If sigHost is specified for the *signer* property specified, then s is a PKCS #7 structure which includes the data that was returned by the StartAuthenticate command. If cmcmac or cmac are specified for the *signer* property specified, then *signatureKey* must refer to a key loaded with the key usage of key attribute is M0 to M8.
key Specifies the name of key being deleted. if this property is omitted. all keys are deleted.

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "accessDenied"`	string	
`}`		

Properties
completionCode The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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

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

8.3.7 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	
`"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 canceled.
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	
`}`		

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

Properties
completionCode The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
errorCode Specifies the error code if applicable. The following values are possible: • `noRSAKeyPair` - The device does not have a private key. • `accessDenied` - The device 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, as one of the following: • `na` - No signature algorithm used, no signature will be provided in signature, the data item may still be exported. • `rsassaPkcs1V15` - RSASSA-PKCS1-v1.5 algorithm used. • `rsassaPss` - RSASSA-PSS algorithm used.
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

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

8.3.8 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 canceled.
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. The following values are possible:

- `rsaPrivate` - Key is used as a private key for RSA decryption.
- `rsaPrivateSign` - Key is used as a private key for RSA Signature generation. Only data generated within the device can be signed.

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. The following values are possible:

- `default` - The device will decide the exponent.
- `exponent1` - Exponent of 21+1 (3).
- `exponent4` - Exponent of 24+1 (17).
- `exponent16` - Exponent of 216+1 (65537).

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

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "accessDenied"`	string	
`}`		

Properties

completionCode

The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription

If included, this contains additional vendor dependent information to assist with problem resolution.

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.
- `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

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

8.3.9 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 allows an application 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

timeout
Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be canceled.
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, as one of the following:

- `na` - No signature will be provided in selfSignature or signature. The requested item may still be exported.
- `rsassaPkcs1V15` - RSASSA-PKCS1-v1.5 algorithm used.
- `rsassaPss` - RSASSA-PSS algorithm used.

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	

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

Payload	Type	Required
`"errorCode": "noRSAKeyPair",`	string	
`"value": Add example to YAML,`	string	
`"selfSignature": Add example to YAML,`	string	
`"signature": Add example to YAML`	string	
`}`		

Properties

completionCode

The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription

If included, this contains additional vendor dependent information to assist with problem resolution.

errorCode

Specifies the error code if applicable. The following values are possible:

- `noRSAKeyPair` - The device does not have a private key.
- `accessDenied` - The device 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

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

8.3.10 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 canceled.

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. The following values are possible:

- `enckey` - The corresponding encryption key is to be returned.
- `verificationkey` - The corresponding verification key is to be returned.
- `hostkey` - The host public key is to be returned.

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "accessDenied",`	string	
`"certificate": Add example to YAML`	string	
`}`		

Properties

completionCode

The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription

If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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

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

8.3.11 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 application 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 canceled.
default: 0

replaceCertificate
The Base64 encoded PKCS # 7 message that will replace the current Certificate Authority. 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
The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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 The Base64 encoded PKCS #7 using a Digested-data content type. The digest parameter should contain the thumb print value.

Event Messages

None

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

8.3.12 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 end 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 canceled.
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
The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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.

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

Properties
randomItem The Base64 encoded randomly generated number created by the encryptor. 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

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

8.3.13 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 canceled.
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. The following values are possible:

- `self` - The key check value (KCV) is created by an encryption of the key with itself. For the description refer to the kcvSelf literal described in the keyCheckModes.
- `zero` - The key check value (KCV) is created by encrypting a zero value with the key. Unless otherwise specified, ECB encryption is used The encryption algorithm used (i.e. DES, 3DES, AES) is determined by the type of key used to generate the KCV.

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
The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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 Base64 encoded key check value data that can be used for verification of the key.

Event Messages

None

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

8.3.14 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 canceled.

default: 0

loadOption

Specifies the method to use to load the certificate. The following values are possible:

- `newHost` - Load a new Host certificate, where one has not already been loaded.
- `replaceHost` - Replace (or rebind) the device to a new Host certificate, where the new Host certificate is signed by *signer*.

signer

Specifies the signer of the certificate to be loaded. The following values are possible:

- `certHost` - The certificate to be loaded is signed by the current Host. Cannot be combined with newHost.
- `ca` - The certificate to be loaded is signed by the Certificate Authority (CA).
- `hl` - The certificate to be loaded is signed by the Higher Level (HL) Authority.

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	

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

Payload	Type	Required
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "accessDenied",`	string	
`"rsaKeyCheckMode": "none",`	string	
`"rsaData": Add example to YAML`	string	
`}`		

Properties
completionCode The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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. The following values are possible: • `none` - No check value is returned in *rsaData* property. • `sha1` - The *rsaData* property contains a sha-1 digest of the public key. • `sha256` - The *rsaData* contains a sha-256 digest of the public key.
rsaData The Base64 encoded PKCS #7 structure using a Digested-data content type. 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

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

8.3.15 KeyManagement.StartAuthenticate

This command is used to retrieve the data that needs to be signed and hence provided to the command with the *authenticate* property in order to perform an authenticated action on the device. If this command returns data to be signed then the *authenticate* property must be used to the command required authenticated action. Any attempt to call the command without the *authenticate* property set, if authentication is required, shall complete with a completionCode authorisationRequired.

Command Message

Payload	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"command": {`	object	
`"deleteKey": {`	object	
`"key": Add example to YAML`	string	
`},`		
`"initialization": {`	object	
`}`		
`}`		
`}`		

Properties

timeout
Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be canceled.
default: 0

command
The command and the input parameters to which authentication is being applied. The possible command is one of:

- `deleteKey` - Delete a key with authentication.
- `initialization` - Initialize encryption module with authentication.

command/deleteKey
This command can be used to delete a key with authentication. Details of KeyManagement.DeleteKey command.

command/deleteKey/key
Specifies the name of key being deleted. if this property is omitted. all keys are deleted.

command/initialization
This command can be used to initialize encryption module with authentication. Details of KeyManagement.Initialization command.

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	

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

Payload	Type	Required
`"errorDescription": Add example to YAML,`	string	
`"dataToSign": Add example to YAML,`	string	
`"signers": "none"`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

dataToSign
The Base64 encoded 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 property is omitted and the command result is success.

signers
Specifies the signer of the data, with one of the following values:

- `none` - Authentication is not required.
- `certhost` - The data is signed by the current Host, using the RSA certificate-based scheme.
- `sigHost` - The data is signed by the current Host, using the RSA signature-based scheme.
- `ca` - The data is signed by the Certificate Authority (CA).
- `hl` - The data is signed by the Higher Level (HL) Authority.
- `cbcmac` - A MAC is calculated over the data using *key* property and the CBC MAC algorithm.
- `cmac` - A MAC is calculated over the data using *key* and the CMAC algorithm.
- `reserved1` - Reserved for a vendor-defined signing method.
- `reserved2` - Reserved for a vendor-defined signing method.
- `reserved3` - Reserved for a vendor-defined signing method.

Event Messages

None

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

8.4 Unsolicited Messages

8.4.1 KeyManagement.InitializedEvent

This is generated when a KeyManagement.Initialization command completed successfully.

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

8.4.2 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 the errorCode property.

Payload	Type	Required
`{`		
`"keyName": Add example to YAML,`	string	Yes
`"errorCode": "keyNotFound"`	string	Yes
`}`		

Properties

keyName
Specifies the name of the key that caused the error.

errorCode
Specifies the type of illegal key access that occurred The following values are possible:

- `keyNotFound` - The specified key was not loaded or attempting to delete a non-existent key.
- `keyNoValue` - The specified key is not loaded.
- `useViolation` - The specified use is not supported by this key.
- `algorithmNotSupp` - The specified algorithm is not supported by this key.

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

8.4.3 KeyManagement.CertificateChangeEvent

This event indicates that the certificate module state has changed from Primary to Secondary.

Payload	Type	Required
`{`		
`"certificateChange": "secondary"`	string	Yes
`}`		

Properties

certificateChange

Specifies change of the certificate state inside of the KeyManagement. The following values are possible:

- `secondary` - The certificate state of the encryptor is now Secondary and Primary Certificates will no longer be accepted.

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

9. Crypto Interface

This chapter defines the Crypto interface functionality and messages.

9.1 Summary

TODO

9.2 General Information

9.2.1 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.

Default Key Name

The DUKPT IPEK key is given a fixed name so multi-vendor applications 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).

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

9.3 Command Messages

9.3.1 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 canceled.
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
The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

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

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

9.3.2 Crypto.CryptoData

The input data is either encrypted or decrypted using the specified or selected encryption mode. The available modes are defined in the cryptAttributes capability. Applications can use an alternative padding method by pre-formatting the data passed and combining this with the standard padding method. The *startValue* (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 *startValue* and *startValueKey* 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": {`	object	
`"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 canceled.
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 omitted 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 Base64 encoded initialization vector for CBC / CFB encryption. If this property and *startValueKey* are both omitted the default value for CBC / CFB is all zeroes.

padding
Specifies the padding character. The valid range is 0 to 255.
Property value constraints:

```
minimum: 0
maximum: 255
```

cryptData
The Base64 encoded data to be encrypted or decrypted.

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

Properties
cryptoAttributes This parameter specifies the encryption algorithm, cryptographic method, and mode to be used for this command. For a list of valid values see cryptoAttributes capability. The values specified must be compatible with the key identified by Key.
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. • `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 The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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 algorithm is not supported. • `cryptoMethodNotSupported` - The cryptographic method specified by *cryptoMethod* is not supported.
cryptData The Base64 encoded encrypted or decrypted data.

Event Messages

- Crypto.DUKPTKSNEvent

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

9.3.3 Crypto.GenerateAuthentication

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. Applications 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. The available attributes are defined in the authenticationAttribute command.

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	
`"compression": false,`	boolean	
`"authenticationData": Add example to YAML,`	string	
`"authenticationAttribute": {`	object	
`"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 canceled.

default: 0

key

Specifies the name of the stored key. The key must have modeOfUse 'C', 'G', 'S' or 'T'.

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.

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

Properties

padding
Specifies the padding character. The valid range is 0 to 255.
Property value constraints:
`minimum: 0`
`maximum: 255`

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 Base64 encoded data to be MACed, or signed.
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 authenticationAttributes property. The values specified must be compatible with the key identified by Key.
authenticationAttribute/cryptoMethod Specifies the cryptographic method cryptoMethod supported. command. For asymmetric signature verification methods (Specified key is key usage '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 the specified key is any of the MAC usages (i.e. 'M1'), then this property can be omitted.
authenticationAttribute/hashAlgorithm For asymmetric signature verification methods (Specified key is key usage 'S0', 'S1', or 'S2'), this can be one of the following values to be used. If the specified key is any of the MAC usages (i.e. 'M1'), then this property can be omitted. • `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 The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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 algorithm 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

- Crypto.DUKPTKSNEvent

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

9.3.4 Crypto.VerifyAuthentication

The authentication data is verified using the specified mode. The available modes are defined in the verifyAttributes 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. Applications can use an alternative padding method by pre-formatting the data passed and combining this with the standard padding method. The *startValue* (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 *startValue* and *startValueKey* 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	
`"compression": false,`	boolean	
`"authenticationData": Add example to YAML,`	string	
`"verifyData": Add example to YAML,`	string	
`"verifyAttributes": {`	object	
`"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 canceled.
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.

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

Properties

padding
Specifies the padding character. The valid range is 0 to 255.
Property value constraints:
`minimum: 0`
`maximum: 255`

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 Base64 encoded *authenticationData* and compare with *verifyData* property.
verifyData The Base64 encoded authentication data to be verified by MAC or signature.
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 verifyAttributes property. The values specified must be compatible with the key identified by Key.
verifyAttributes/cryptoMethod Specifies the cryptographic method cryptoMethod supported. For asymmetric signature verification methods (Specified key is key usage '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 the specified key is any of the MAC usages (i.e. 'M1'), then this property can be omitted.
verifyAttributes/hashAlgorithm For asymmetric signature verification methods (Specified key is key usage 'S0', 'S1', or 'S2'), this can be one of the following values to be used. If the specified key is any of the MAC usages (i.e. 'M1'), then this property can be omitted. • `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 The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

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

Properties
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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 algorithm 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

- Crypto.DUKPTKSNEvent

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

9.3.5 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	
`}`		

Properties

timeout
Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be canceled.
default: 0

hashAlgorithm
Specifies which hash algorithm should be used to calculate the hash. See the verifyAttributes capability for valid algorithms. The following values are possible:

- `sha1` - The SHA-1 digest algorithm.
- `sha256` - The SHA-256 digest algorithm, as defined in ISO/IEC 10118-3:2004 and FIPS 180-2.

digestInput
Contains Base64 encoded the length and the data to be hashed.

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
The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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 Base64 encoded length and the data containing the calculated hash.

Event Messages

None

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

9.4 Event Messages

9.4.1 Crypto.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	Yes
`"ksn": Add example to YAML`	string	Yes
`}`		

Properties
key Specifies the name of the DUKPT Key derivation key.
ksn The Base64 encoded KSN.

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

9.5 Unsolicited Messages

9.5.1 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 the errorCode property.

Payload	Type	Required
`{`		
`"keyName": Add example to YAML,`	string	Yes
`"errorCode": "keyNotFound"`	string	Yes
`}`		

Properties

keyName
Specifies the name of the key that caused the error.

errorCode
Specifies the type of illegal key access that occurred The following values are possible:

- `keyNotFound` - The specified key was not loaded or attempting to delete a non-existent key.
- `keyNoValue` - The specified key is not loaded.
- `useViolation` - The specified use is not supported by this key.
- `algorithmNotSupp` - The specified algorithm is not supported by this key.

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

10. Keyboard Interface

This chapter defines the Keyboard interface functionality and messages.

10.1 Summary

This section describes the general interface for the following functions:

- Entering Personal Identification Numbers (PINs)
- Clear text data handling
- Function key handlingIf the 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.

10.2 General Information

10.2.1 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 application 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

-

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

```
Those which activate touch areas defined by the application.<br>
```

-

```
Those which activate a random variation of touch areas defined by the application.<br>
```

The Service Provider, when reporting its capabilities, reports the absolute position of the ETS in Windows desktop coordinates. This allows the application 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 zero through nine (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 zero through nine and not a through 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. An application 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 application has to be trusted.

Depending upon the type of device, the application 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 application 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.

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

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 application 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.

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

10.2.2 Layout

The following table defines the possible size and position values that apply to each frame type.

| Frame Type FRAME KEY | | :--- | --- | --- | --- | | frameXSize | frameYSize | framexPos | frameyPos | xSize | ySize | xPos | yPos | | Physical Keys on EPP | 0 | 0 | 0 | 0 | 1 to 1000[1] | 1 to 1000[2] | 0 to 999[3] | 0 to 999[4] | | Touch Keys on ETS | > 0 | > 0 | >= 0 | >= 0 | 0 to (frameXSize - xPos) | 0 to (frameYSize - yPos) | 0 to frameXSize | 0 to frameYSize | | Physical Keys on Left Boundary of Screen | 0 | > 0 | 0 | 0 | 0 | 0 to (frameYSize - yPos) | 0 | 0 to frameYSize | | | | | | | | | | | Physical Keys on Right Boundary of Screen | 0 | > 0 | > 0 | 0 | 0 | 0 to (frameYSize - yPos) | frameXSize | 0 to frameYSize | | Physical Keys on Top Boundary of Screen | > 0 | 0 | 0 | 0 | 0 to(frameXSize - xPos) | 0 | 0 to frameXSize | 0 | | Physical Keys on Bottom Boundary of Screen | > 0 | 0 | 0 | > 0 | 0 to (frameXSize - xPos) | 0 | 0 to frameXSize | frameYSize |

[1]: 1 is the smallest possible size and 1000 is the full width of the frame
[2]: 1 is the smallest possible size and 1000 is the full height of the frame
[3]: 0 is the left edge and 999 is the right edge of the frame
[4]: 0 is the top edge and 999 is the bottom edge of the frame

The following diagram shows an example configuration consisting of an EPP and Physical FDKs to the left and right of the screen. 3 frames contain the Physical Keys.

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

10.2.3 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 application 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.

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

HOST → PIN

SecureKeyDetail

Display Keyboard Layout

SecureKeyEntry (Part 1)
[part 1 master encryption key values are manually entered on the PinPad]

Verify KCV
[verify part 1 master key KCV as returned with the output data of the SecureKeyEntry command]

KeyManagement.ImportKey
[part 1 master encryption key values loaded with the secure key entry command]

```
{
  "header": {
    "type": "command",
    "name": "KeyManagement.ImportKey",
    "requestId": 12345
  },
  "payload" : {
    "key": "MasterKey",
    "keyAttributes": {
      "keyUsage": "K0",
      "algorithm": "T",
      "modeOfUse": "B"
    "constructing": true
  }
}
```

SecureKeyEntry
[part 2 crypt master encryption key values are manually entered on the PinPad]

Verify KCV
[verify part 2 master key KCV as returned with the output data of the SecurekeyEntry command]

KeyManagement.ImportKey
[part 2 master encryption key values loaded with the secure key entry command]

```
{
  "header": {
    "type": "command",
    "name": "KeyManagement.ImportKey",
    "requestId": 12346
  },
  "payload" : {
    "key": "MasterKey",
    "keyAttributes": {
      "keyUsage": "K0",
      "algorithm": "T",
      "modeOfUse": "B"
    "constructing": true
  }
}
```

KeyManagement.ImportKey
[activates the full master encryption key]

```
{
  "header": {
    "type": "command",
    "name": "KeyManagement.ImportKey",
    "requestId": 12347
  },
  "payload" : {
    "key": "MasterKey",
    "keyAttributes": {
      "keyUsage": "K0",
      "algorithm": "T",
      "modeOfUse": "B"
    }
  }
}
```

Verify KCV
[verify full master key KCV as returned with the output data of the SecurekeyEntry command]


```
{
  "header": {
    "type": "completion",
    "name": "KeyManagement.ImportKey",
    "requestId": 12347
  },
  "payload" : {
    "verificationData": "MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTIzNDU2Nzg5MDEyMzQ1Njc4OQ==",
    "verifyAttributes": {
      "00": {
        "T": {
         "V": {
            "cryptoMethod": "kcvZero"
         }
        }
    },
    "keyLength": 128
  }
}
```


HOST → PIN

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

10.3 Command Messages

10.3.1 Keyboard.GetLayout

This command allows an application 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 canceled.

default: 0

entryMode

Specifies entry mode to be returned. If this property is omitted, all supported layouts will be returned. The following values are possible:

- `data` - Get the layout for the DataEntry command.
- `pin` - Get the layout for the PinEntry command.
- `secure` - Get the layout for the SecureKeyEntry command.

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "modeNotSupported",`	string	
`"layout": {`	object	
`"data": [{`	array (object)	
`"xPos": 0,`	integer	Yes
`"yPos": 0,`	integer	Yes
`"xSize": 0,`	integer	Yes

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

Payload	Type	Required
"ySize": 0,	integer	Yes
"float": {	object	
"x": false,	boolean	
"y": false	boolean	
},		
"keys": [{	array (object)	
"xPos": 0,	integer	Yes
"yPos": 0,	integer	Yes
"xSize": 1,	integer	Yes
"ySize": 1,	integer	Yes
"key": "one",	string	
"shiftKey": "a"	string	
}]		
}],		
"pin": [{	array (object)	
"xPos": 0,	integer	Yes
"yPos": 0,	integer	Yes
"xSize": 0,	integer	Yes
"ySize": 0,	integer	Yes
"float": {	object	
"x": false,	boolean	
"y": false	boolean	
},		
"keys": [{	array (object)	
"xPos": 0,	integer	Yes
"yPos": 0,	integer	Yes
"xSize": 1,	integer	Yes
"ySize": 1,	integer	Yes
"key": "one",	string	
"shiftKey": "a"	string	
}]		
}],		
"secure": [{	array (object)	
"xPos": 0,	integer	Yes
"yPos": 0,	integer	Yes
"xSize": 0,	integer	Yes
"ySize": 0,	integer	Yes

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

Payload	Type	Required
`"float": {`	object	
`"x": false,`	boolean	
`"y": false`	boolean	
`},`		
`"keys": [{`	array (object)	
`"xPos": 0,`	integer	Yes
`"yPos": 0,`	integer	Yes
`"xSize": 1,`	integer	Yes
`"ySize": 1,`	integer	Yes
`"key": "one",`	string	
`"shiftKey": "a"`	string	
`}]`		
`}]`		
`}`		
`}`		

Properties

completionCode
The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

errorCode
Specifies the error code if applicable. The following values are possible:

- `modeNotSupported` - The specified entry mode is not supported.

layout
Return supported layouts specified by the *entryMode* property.

layout/data
The layout for the DataEntry command.

layout/data/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

layout/data/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

layout/data/xSize
For ETS, specifies the width of the frame. For all other device types, this value is ignored

layout/data/ySize
For ETS, specifies the height of the frame. For all other device types, this value is ignored

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

Properties
layout/data/float Specifies if the device can float the touch keyboards
layout/data/float/x Specifies that the device will randomly shift the layout in a horizontal direction default: false
layout/data/float/y Specifies that the device will randomly shift the layout in a vertical direction default: false
layout/data/keys Defining details of the keys in the keyboard.

layout/data/keys/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.
Property value constraints:
`minimum: 0`
`maximum: 999`

layout/data/keys/yPos
Specifies the position of the top left corner of the Function Key (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.
Property value constraints:
`minimum: 0`
`maximum: 999`

layout/data/keys/xSize
Specifies the Function Key (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.
Property value constraints:
`minimum: 1`
`maximum: 1000`

layout/data/keys/ySize
Specifies the Function Key (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.
Property value constraints:
`minimum: 1`
`maximum: 1000`

layout/data/keys/key Specifies the Function Key associated with the physical area in non-shifted mode. This property is not required if the device doesn't support. Property value constraints: `pattern: ^(one|two|three|four|five|six|seven|eight|nine|[a-f]|enter|cancel|clear|backspace|help|decPoint|shift|doubleZero|tripleZero)$\|^fdk(0[1-9]\|[12][0-9]\|3[0-2])$|.+`

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

Properties
layout/data/keys/shiftKey Specifies the Function Key associated with the physical key in shifted mode. This property is not required if the device doesn't support. Property value constraints: `pattern: ^(one|two|three|four|five|six|seven|eight|nine|[a-f]|enter|cancel|clear|backspace|help|decPoint|shift|doubleZero|tripleZero)$\|^fdk(0[1-9]\|[12][0-9]\|3[0-2])$|.+`
layout/pin The layout for the PinEntry command.
layout/pin/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
layout/pin/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
layout/pin/xSize For ETS, specifies the width of the frame. For all other device types, this value is ignored
layout/pin/ySize For ETS, specifies the height of the frame. For all other device types, this value is ignored
layout/pin/float Specifies if the device can float the touch keyboards
layout/pin/float/x Specifies that the device will randomly shift the layout in a horizontal direction default: false
layout/pin/float/y Specifies that the device will randomly shift the layout in a vertical direction default: false
layout/pin/keys Defining details of the keys in the keyboard.

layout/pin/keys/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.
Property value constraints:
`minimum: 0`
`maximum: 999`

layout/pin/keys/yPos
Specifies the position of the top left corner of the Function Key (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.
Property value constraints:
`minimum: 0`
`maximum: 999`

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

Properties

layout/pin/keys/xSize
Specifies the Function Key (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.
Property value constraints:
`minimum: 1`
`maximum: 1000`

layout/pin/keys/ySize
Specifies the Function Key (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.
Property value constraints:
`minimum: 1`
`maximum: 1000`

layout/pin/keys/key Specifies the Function Key associated with the physical area in non-shifted mode. This property is not required if the device doesn't support. Property value constraints: `pattern: ^(one|two|three|four|five|six|seven|eight|nine|[a-f]|enter|cancel|clear|backspace|help|decPoint|shift|doubleZero|tripleZero)$\|^fdk(0[1-9]\|[12][0-9]\|3[0-2])$|.+`
layout/pin/keys/shiftKey Specifies the Function Key associated with the physical key in shifted mode. This property is not required if the device doesn't support. Property value constraints: `pattern: ^(one|two|three|four|five|six|seven|eight|nine|[a-f]|enter|cancel|clear|backspace|help|decPoint|shift|doubleZero|tripleZero)$\|^fdk(0[1-9]\|[12][0-9]\|3[0-2])$|.+`
layout/secure The layout for the SecureKeyEntry command.
layout/secure/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
layout/secure/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
layout/secure/xSize For ETS, specifies the width of the frame. For all other device types, this value is ignored
layout/secure/ySize For ETS, specifies the height of the frame. For all other device types, this value is ignored
layout/secure/float Specifies if the device can float the touch keyboards
layout/secure/float/x Specifies that the device will randomly shift the layout in a horizontal direction default: false

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

Properties
layout/secure/float/y Specifies that the device will randomly shift the layout in a vertical direction default: false
layout/secure/keys Defining details of the keys in the keyboard.

layout/secure/keys/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.
Property value constraints:
`minimum: 0`
`maximum: 999`

layout/secure/keys/yPos
Specifies the position of the top left corner of the Function Key (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.
Property value constraints:
`minimum: 0`
`maximum: 999`

layout/secure/keys/xSize
Specifies the Function Key (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.
Property value constraints:
`minimum: 1`
`maximum: 1000`

layout/secure/keys/ySize
Specifies the Function Key (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.
Property value constraints:
`minimum: 1`
`maximum: 1000`

layout/secure/keys/key Specifies the Function Key associated with the physical area in non-shifted mode. This property is not required if the device doesn't support. Property value constraints: `pattern: ^(one|two|three|four|five|six|seven|eight|nine|[a-f]|enter|cancel|clear|backspace|help|decPoint|shift|doubleZero|tripleZero)$\|^fdk(0[1-9]\|[12][0-9]\|3[0-2])$|.+`
layout/secure/keys/shiftKey Specifies the Function Key associated with the physical key in shifted mode. This property is not required if the device doesn't support. Property value constraints: `pattern: ^(one|two|three|four|five|six|seven|eight|nine|[a-f]|enter|cancel|clear|backspace|help|decPoint|shift|doubleZero|tripleZero)$\|^fdk(0[1-9]\|[12][0-9]\|3[0-2])$|.+`

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

Event Messages

None

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

10.3.2 Keyboard.PinEntry

This function stores the PIN entry via the device. From the point this function is invoked, PIN digit entries are not passed to the application. For each PIN digit, or any other active key entered, an execute notification event Keyboard.KeyEvent is sent in order to allow an application to perform the appropriate display action (i.e. when the PIN pad has no integrated display). The application 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 Keyboard is ready for the user to start entering data.

Some devices do not inform the application 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 Function Descriptor Keys(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 application sets terminate property. In the event that terminate property is set to false or omitted all active keys and *maxLen* is zero, the command will not terminate and the application must issue a Common.Cancel command.

If active the 'cancel' and 'clear' keys will cause the PIN buffer to be cleared. The backspace 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 Common.Cancel command 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 device cannot ignore the enter key then the command will complete normally. To handle these types of devices the application should use the output parameter digits property to check that sufficient digits have been entered. The application should then get the user to re-enter their PIN with the correct number of digits.

If the application 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 application 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": Add example to YAML,`	string	
`"activeKeys": {`	object	

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

Payload	Type	Required
`"one": {`	object	
`"terminate": false`	boolean	
`},`		
`"backspace": {`	object	
`See one properties.`		
`},`		
`"oem1": {`	object	
`See one properties.`		
`},`		
`"res6": {`	object	
`See oem1 properties.`		
`}`		
`}`		
`}`		

Properties

timeout

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

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.

Property value constraints:

```
minLength: 1
maxLength: 1
```

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

Properties
activeKeys Specifies all Function Keys which are active during the execution of the command. This should be the complete set or a subset of the keys returned in the payload of the Keyboard.GetLayout command. The following standard names are defined: • `zero` - Numeric digit 0 • `one` - Numeric digit 1 • `two` - Numeric digit 2 • `three` - Numeric digit 3 • `four` - Numeric digit 4 • `five` - Numeric digit 5 • `six` - Numeric digit 6 • `seven` - Numeric digit 7 • `eight` - Numeric digit 8 • `nine` - Numeric digit 9 • `[a-f]` - Hex digit A to F for secure key entry • `enter` - Enter • `cancel` - Cancel • `clear` - Clear • `backspace` - Backspace • `help` - Help • `decPoint` - Decimal point • `shift` - Shift key used during hex entry • `doubleZero` - 00 • `tripleZero` - 000 • `fdk[01-32]` - 32 FDK keys Additional non standard key names are also allowed.
activeKeys/one (example name) A standard key name. Property name constraints: `pattern: ^(one|two|three|four|five|six|seven|eight|nine|[a-f]|enter|cancel|clear|backspace|help|decPoint|shift|doubleZero|tripleZero)$\|^fdk(0[1-9]\|[12][0-9]\|3[0-2])$`
activeKeys/one/terminate The key is a terminate key. default: false
activeKeys/oem1 (example name) A non standard key name.

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "keyInvalid",`	string	

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

Payload	Type	Required
`"digits": 0,`	integer	
`"completion": "auto"`	string	
`}`		

Properties

completionCode

The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription

If included, this contains additional vendor dependent information to assist with problem resolution.

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 property is invalid or greater than the maximum PIN length property 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

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

10.3.3 Keyboard.DataEntry

This function is used to return keystrokes entered by the user. It will automatically set the PIN pad to echo characters on the display if there is a display. For each keystroke a notification event Keyboard.KeyEvent is sent in order to allow an application to perform the appropriate display action (i.e. when the PIN pad has no integrated display).

The Keyboard.EnterDataEvent will be generated when the PIN pad is ready for the user to start entering data.

When the maximum number of digits is entered and the *autoEnd* property is true, or a terminate key is pressed after the minimum number of digits is entered, the command completes. If the key is a terminator key and is pressed, the command will complete successfully even if the minimum number of digits has not been entered.

Terminating Function Descriptor Keys(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 application sets terminate property. In the event that terminate property is set to false or omitted all active keys and *maxLen* is zero, the command will not terminate and the application must issue a Common.Cancel command.

If *maxLen* has been met and *autoEnd* is set to False, then all keys or FDKs that add data to the contents of the output parameter will automatically be disabled. If the CLEAR or BACKSPACE key is pressed to reduce the number of entered keys below *maxLen*, the same keys will be re-enabled.

Where applications want direct control of the data entry and the key interpretation, *maxLen* can be set to zero allowing the application to provide tracking and counting of key presses until a terminate key is pressed or Common.Cancel has been issued.

The following keys may affect the contents of the output parameter but are not returned in it: 'enter' 'cancel' 'clear' 'backspace' The 'cancel' and 'clear' keys will cause the output buffer to be cleared. The 'backspace' key will cause the last key in the buffer to be removed.

Terminating keys have to be active keys to operate.

It is the responsibility of the application to identify the mapping between the FDK code and the physical location of the FDK.

Command Message

Payload	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"maxLen": 0,`	integer	
`"autoEnd": false,`	boolean	
`"activeKeys": {`	object	
`"one": {`	object	
`"terminate": false`	boolean	
`},`		
`"backspace": {`	object	
`See one properties.`		
`},`		
`"oem1": {`	object	

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

Payload	Type	Required
`See one properties.`		
`},`		
`"res6": {`	object	
`See oem1 properties.`		
`}`		
`}`		
`}`		

Properties

timeout

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

default: 0

maxLen

Specifies the maximum number of digits which can be returned to the application 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.

activeKeys

Specifies all Function Keys which are active during the execution of the command. This should be the complete set or a subset of the keys returned in the payload of the Keyboard.GetLayout command.

The following standard names are defined:

- `zero` - Numeric digit 0
- `one` - Numeric digit 1
- `two` - Numeric digit 2
- `three` - Numeric digit 3
- `four` - Numeric digit 4
- `five` - Numeric digit 5
- `six` - Numeric digit 6
- `seven` - Numeric digit 7
- `eight` - Numeric digit 8
- `nine` - Numeric digit 9
- `[a-f]` - Hex digit A to F for secure key entry
- `enter` - Enter
- `cancel` - Cancel
- `clear` - Clear
- `backspace` - Backspace
- `help` - Help
- `decPoint` - Decimal point
- `shift` - Shift key used during hex entry
- `doubleZero` - 00
- `tripleZero` - 000
- `fdk[01-32]` - 32 FDK keys

Additional non standard key names are also allowed.

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

Properties
activeKeys/one (example name) A standard key name. Property name constraints: `pattern: ^(one|two|three|four|five|six|seven|eight|nine|[a-f]|enter|cancel|clear|backspace|help|decPoint|shift|doubleZero|tripleZero)$\|^fdk(0[1-9]\|[12][0-9]\|3[0-2])$`
activeKeys/one/terminate The key is a terminate key. default: false
activeKeys/oem1 (example name) A non standard key name.

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "keyInvalid",`	string	
`"keys": 0,`	integer	
`"pinKeys": [{`	array (object)	
`"completion": "auto",`	string	
`"digit": "five"`	string	
`}],`		
`"completion": "auto"`	string	
`}`		

Properties
completionCode The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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 contains the keys entered by the user

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

Properties
pinKeys/completion Specifies the reason for completion of the entry. The following values are possible: • `auto` - The command terminated automatically, because maximum length was reached. • `enter` - The ENTER Function Key was pressed as terminating key. • `cancel` - The CANCEL Function Key was pressed as terminating key. • `continue` - A function key was pressed and input may continue unless the command completes (this value is only used in the execute event Keyboard.KeyEvent and in the array of keys in the completion of Keyboard.DataEntry). • `clear` - The clear function Key was pressed as terminating key and the previous input is cleared. • `backspace` - The last input digit was cleared and the key was pressed as terminating key. • `fdk` - Indicates input is terminated only if the FDK pressed was set to be a terminating FDK. • `help` - The HELP Function Key was pressed as terminating key. • `fk` - A Function Key (FK) other than ENTER, CLEAR, CANCEL, BACKSPACE, HELP was pressed as terminating key. • `contFdk` - An Function Descriptor Key (FDK) was pressed and input may continue unless the command completes (this value is only used in the Keyboard.KeyEvent and in the array of keys in the completion of Keyboard.DataEntry).
pinKeys/digit Specifies the digit entered by the user. When working in encryption mode or secure key entry mode (Keyboard.PinEntry and Keyboard.SecureKeyEntry), this property is omitted for the function keys 'one' to 'nine' and 'a' to 'f'. Otherwise, for each key pressed, the corresponding key value is stored in this property. Property value constraints: `pattern: ^(one|two|three|four|five|six|seven|eight|nine|[a-f]|enter|cancel|clear|backspace|help|decPoint|shift|doubleZero|tripleZero)$\|^fdk(0[1-9]\|[12][0-9]\|3[0-2])$|.+`
completion Specifies the reason for completion of the entry.

Event Messages

- Keyboard.KeyEvent
- Keyboard.EnterDataEvent
- Keyboard.LayoutEvent

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

10.3.4 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 canceled.
default: 0

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

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

10.3.5 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 ('zero' to 'nine' and 'a' to 'f') are not passed to the application. 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 an application to perform the appropriate display action (i.e. when the device has no integrated display). When an encryption key digit is entered the application 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 Keyboard.GetLayout 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 *keyLen*, the same keys will be reenabled.

Terminating keys have to be active keys to operate.

If an Function Descriptor Key(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 Keyboard.KeyEvent, applications 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 application must include the 'fdk01' FDK code in the *activeKeys* parameter (if the clear functionality is required). In addition when this FDK is pressed the Keyboard.KeyEvent will contain the 'fdk01' mask value in the digit property. The application 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 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 application must set the *autoEnd* flag to TRUE and wait for the command to auto-complete. Applications 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	
`"activeKeys": {`	object	
`"one": {`	object	
`"terminate": false`	boolean	

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

Payload	Type	Required
`},`		
`"backspace": {`	object	
`See one properties.`		
`},`		
`"oem1": {`	object	
`See one properties.`		
`},`		
`"res6": {`	object	
`See oem1 properties.`		
`}`		
`},`		
`"verificationType": "self",`	string	
`"cryptoMethod": "des"`	string	
`}`		

Properties

timeout

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

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

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

Properties
activeKeys Specifies all Function Keys which are active during the execution of the command. This should be the complete set or a subset of the keys returned in the payload of the Keyboard.GetLayout command. This should include 'zero' to 'nine' and 'a' to 'f' for all modes of secure key entry, but should also include 'shift' on shift based systems. The 'doubleZero', 'tripleZero' and 'decPoint' function keys must not be included in the list of active or terminate keys. For FDKs which must terminate the execution of the command. This should include the FDKs associated with Cancel and Enter. The following standard names are defined: • `zero` - Numeric digit 0 • `one` - Numeric digit 1 • `two` - Numeric digit 2 • `three` - Numeric digit 3 • `four` - Numeric digit 4 • `five` - Numeric digit 5 • `six` - Numeric digit 6 • `seven` - Numeric digit 7 • `eight` - Numeric digit 8 • `nine` - Numeric digit 9 • `[a-f]` - Hex digit A to F for secure key entry • `enter` - Enter • `cancel` - Cancel • `clear` - Clear • `backspace` - Backspace • `help` - Help • `decPoint` - Decimal point • `shift` - Shift key used during hex entry • `doubleZero` - 00 • `tripleZero` - 000 • `fdk[01-32]` - 32 FDK keys Additional non standard key names are also allowed for terminating the execution of the command.
activeKeys/one (example name) A standard key name. Property name constraints: `pattern: ^(one|two|three|four|five|six|seven|eight|nine|[a-f]|enter|cancel|clear|backspace|help|decPoint|shift|doubleZero|tripleZero)$\|^fdk(0[1-9]\|[12][0-9]\|3[0-2])$`
activeKeys/one/terminate The key is a terminate key. default: false
activeKeys/oem1 (example name) A non standard key name.

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

Properties
verificationType Specifies the type of verification to be done on the entered key. The following values are possible: • `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. • `zero` - The key check value is created by encrypting a zero value with the key.
cryptoMethod Specifies the cryptographic method to be used for the verification. If this property is omitted, *keyLen* will determine the cryptographic method used. If *keyLen* is 16, the cryptographic method will be Single DES. If *keyLen* is 32 or 48, the cryptographic method will be Triple DES The following values are possible: • `des` - Single DES • `tripleDes` - Triple DES • `aes` - AES

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 The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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. Applications must ensure all required digits have been entered before trying to store the key.
completion Specifies the reason for completion of the entry. The following values are possible: • `auto` - The command terminated automatically, because maximum length was reached. • `enter` - The ENTER Function Key was pressed as terminating key. • `cancel` - The CANCEL Function Key was pressed as terminating key. • `continue` - A function key was pressed and input may continue unless the command completes (this value is only used in the execute event Keyboard.KeyEvent and in the array of keys in the completion of Keyboard.DataEntry). • `clear` - The clear function Key was pressed as terminating key and the previous input is cleared. • `backspace` - The last input digit was cleared and the key was pressed as terminating key. • `fdk` - Indicates input is terminated only if the FDK pressed was set to be a terminating FDK. • `help` - The HELP Function Key was pressed as terminating key. • `fk` - A Function Key (FK) other than ENTER, CLEAR, CANCEL, BACKSPACE, HELP was pressed as terminating key. • `contFdk` - An Function Descriptor Key (FDK) was pressed and input may continue unless the command completes (this value is only used in the Keyboard.KeyEvent and in the array of keys in the completion of Keyboard.DataEntry).
kcv Contains the key check value data that can be used for verification of the entered key formatted in base 64. This property it omitted 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

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

10.3.6 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 application control of automatic beeping. See autoBeep capability for information.

Command Message

Payload	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"mode": {`	object	
`"active": false,`	boolean	Yes
`"inactive": false`	boolean	Yes
`}`		
`}`		

Properties

timeout
Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be canceled.
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
The completion code.

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

Properties
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

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

10.3.7 Keyboard.DefineLayout

This command allows an application 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
`{`		
`"timeout": 5000,`	integer	
`"layout": {`	object	
`"data": [{`	array (object)	
`"xPos": 0,`	integer	Yes
`"yPos": 0,`	integer	Yes
`"xSize": 0,`	integer	Yes
`"ySize": 0,`	integer	Yes
`"float": {`	object	
`"x": false,`	boolean	
`"y": false`	boolean	
`},`		
`"keys": [{`	array (object)	
`"xPos": 0,`	integer	Yes
`"yPos": 0,`	integer	Yes
`"xSize": 1,`	integer	Yes
`"ySize": 1,`	integer	Yes
`"key": "one",`	string	
`"shiftKey": "a"`	string	
`}]`		
`}],`		
`"pin": [{`	array (object)	
`"xPos": 0,`	integer	Yes
`"yPos": 0,`	integer	Yes
`"xSize": 0,`	integer	Yes

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

Payload	Type	Required
"ySize": 0,	integer	Yes
"float": {	object	
"x": false,	boolean	
"y": false	boolean	
},		
"keys": [{	array (object)	
"xPos": 0,	integer	Yes
"yPos": 0,	integer	Yes
"xSize": 1,	integer	Yes
"ySize": 1,	integer	Yes
"key": "one",	string	
"shiftKey": "a"	string	
}]		
}],		
"secure": [{	array (object)	
"xPos": 0,	integer	Yes
"yPos": 0,	integer	Yes
"xSize": 0,	integer	Yes
"ySize": 0,	integer	Yes
"float": {	object	
"x": false,	boolean	
"y": false	boolean	
},		
"keys": [{	array (object)	
"xPos": 0,	integer	Yes
"yPos": 0,	integer	Yes
"xSize": 1,	integer	Yes
"ySize": 1,	integer	Yes
"key": "one",	string	
"shiftKey": "a"	string	
}]		
}]		
}		
}		

© 2021 CEN 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 canceled. default: 0
layout Specify layouts to define.
layout/data The layout for the DataEntry command.
layout/data/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
layout/data/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
layout/data/xSize For ETS, specifies the width of the frame. For all other device types, this value is ignored
layout/data/ySize For ETS, specifies the height of the frame. For all other device types, this value is ignored
layout/data/float Specifies if the device can float the touch keyboards
layout/data/float/x Specifies that the device will randomly shift the layout in a horizontal direction default: false
layout/data/float/y Specifies that the device will randomly shift the layout in a vertical direction default: false
layout/data/keys Defining details of the keys in the keyboard.

layout/data/keys/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.
Property value constraints:
`minimum: 0`
`maximum: 999`

layout/data/keys/yPos
Specifies the position of the top left corner of the Function Key (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.
Property value constraints:
`minimum: 0`
`maximum: 999`

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

Properties

layout/data/keys/xSize
Specifies the Function Key (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.
Property value constraints:
`minimum: 1`
`maximum: 1000`

layout/data/keys/ySize
Specifies the Function Key (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.
Property value constraints:
`minimum: 1`
`maximum: 1000`

layout/data/keys/key Specifies the Function Key associated with the physical area in non-shifted mode. This property is not required if the device doesn't support. Property value constraints: `pattern: ^(one|two|three|four|five|six|seven|eight|nine|[a-f]|enter|cancel|clear|backspace|help|decPoint|shift|doubleZero|tripleZero)$\|^fdk(0[1-9]\|[12][0-9]\|3[0-2])$|.+`
layout/data/keys/shiftKey Specifies the Function Key associated with the physical key in shifted mode. This property is not required if the device doesn't support. Property value constraints: `pattern: ^(one|two|three|four|five|six|seven|eight|nine|[a-f]|enter|cancel|clear|backspace|help|decPoint|shift|doubleZero|tripleZero)$\|^fdk(0[1-9]\|[12][0-9]\|3[0-2])$|.+`
layout/pin The layout for the PinEntry command.
layout/pin/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
layout/pin/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
layout/pin/xSize For ETS, specifies the width of the frame. For all other device types, this value is ignored
layout/pin/ySize For ETS, specifies the height of the frame. For all other device types, this value is ignored
layout/pin/float Specifies if the device can float the touch keyboards
layout/pin/float/x Specifies that the device will randomly shift the layout in a horizontal direction default: false

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

Properties
layout/pin/float/y Specifies that the device will randomly shift the layout in a vertical direction default: false
layout/pin/keys Defining details of the keys in the keyboard.

layout/pin/keys/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.
Property value constraints:
`minimum: 0`
`maximum: 999`

layout/pin/keys/yPos
Specifies the position of the top left corner of the Function Key (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.
Property value constraints:
`minimum: 0`
`maximum: 999`

layout/pin/keys/xSize
Specifies the Function Key (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.
Property value constraints:
`minimum: 1`
`maximum: 1000`

layout/pin/keys/ySize
Specifies the Function Key (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.
Property value constraints:
`minimum: 1`
`maximum: 1000`

layout/pin/keys/key Specifies the Function Key associated with the physical area in non-shifted mode. This property is not required if the device doesn't support. Property value constraints: `pattern: ^(one|two|three|four|five|six|seven|eight|nine|[a-f]|enter|cancel|clear|backspace|help|decPoint|shift|doubleZero|tripleZero)$\|^fdk(0[1-9]\|[12][0-9]\|3[0-2])$|.+`
layout/pin/keys/shiftKey Specifies the Function Key associated with the physical key in shifted mode. This property is not required if the device doesn't support. Property value constraints: `pattern: ^(one|two|three|four|five|six|seven|eight|nine|[a-f]|enter|cancel|clear|backspace|help|decPoint|shift|doubleZero|tripleZero)$\|^fdk(0[1-9]\|[12][0-9]\|3[0-2])$|.+`

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

Properties
layout/secure The layout for the SecureKeyEntry command.
layout/secure/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
layout/secure/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
layout/secure/xSize For ETS, specifies the width of the frame. For all other device types, this value is ignored
layout/secure/ySize For ETS, specifies the height of the frame. For all other device types, this value is ignored
layout/secure/float Specifies if the device can float the touch keyboards
layout/secure/float/x Specifies that the device will randomly shift the layout in a horizontal direction default: false
layout/secure/float/y Specifies that the device will randomly shift the layout in a vertical direction default: false
layout/secure/keys Defining details of the keys in the keyboard.

layout/secure/keys/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.
Property value constraints:
`minimum: 0`
`maximum: 999`

layout/secure/keys/yPos
Specifies the position of the top left corner of the Function Key (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.
Property value constraints:
`minimum: 0`
`maximum: 999`

layout/secure/keys/xSize
Specifies the Function Key (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.
Property value constraints:
`minimum: 1`
`maximum: 1000`

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

Properties

layout/secure/keys/ySize
Specifies the Function Key (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.
Property value constraints:
`minimum: 1`
`maximum: 1000`

layout/secure/keys/key Specifies the Function Key associated with the physical area in non-shifted mode. This property is not required if the device doesn't support. Property value constraints: `pattern: ^(one|two|three|four|five|six|seven|eight|nine|[a-f]|enter|cancel|clear|backspace|help|decPoint|shift|doubleZero|tripleZero)$\|^fdk(0[1-9]\|[12][0-9]\|3[0-2])$|.+`
layout/secure/keys/shiftKey Specifies the Function Key associated with the physical key in shifted mode. This property is not required if the device doesn't support. Property value constraints: `pattern: ^(one|two|three|four|five|six|seven|eight|nine|[a-f]|enter|cancel|clear|backspace|help|decPoint|shift|doubleZero|tripleZero)$\|^fdk(0[1-9]\|[12][0-9]\|3[0-2])$|.+`

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "modeNotSupported"`	string	
`}`		

Properties
completionCode The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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 *keyType* and values for *fK* and *shiftFK* can only be used once per layout.

Event Messages

None

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

10.4 Event Messages

10.4.1 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 application has to manage the display of the entered digits. It is the responsibility of the application to identify the mapping between the FDK code and the physical location of the FDK.

Payload	Type	Required
`{`		
`"completion": "auto",`	string	
`"digit": "five"`	string	
`}`		

Properties

completion

Specifies the reason for completion of the entry. The following values are possible:

- `auto` - The command terminated automatically, because maximum length was reached.
- `enter` - The ENTER Function Key was pressed as terminating key.
- `cancel` - The CANCEL Function Key was pressed as terminating key.
- `continue` - A function key was pressed and input may continue unless the command completes (this value is only used in the execute event Keyboard.KeyEvent and in the array of keys in the completion of Keyboard.DataEntry).
- `clear` - The clear function Key was pressed as terminating key and the previous input is cleared.
- `backspace` - The last input digit was cleared and the key was pressed as terminating key.
- `fdk` - Indicates input is terminated only if the FDK pressed was set to be a terminating FDK.
- `help` - The HELP Function Key was pressed as terminating key.
- `fk` - A Function Key (FK) other than ENTER, CLEAR, CANCEL, BACKSPACE, HELP was pressed as terminating key.
- `contFdk` - An Function Descriptor Key (FDK) was pressed and input may continue unless the command completes (this value is only used in the Keyboard.KeyEvent and in the array of keys in the completion of Keyboard.DataEntry).

digit

Specifies the digit entered by the user. When working in encryption mode or secure key entry mode (Keyboard.PinEntry and Keyboard.SecureKeyEntry), this property is omitted for the function keys 'one' to 'nine' and 'a' to 'f'. Otherwise, for each key pressed, the corresponding key value is stored in this property.

Property value constraints:

```
pattern: ^(one|two|three|four|five|six|seven|eight|nine|[a-
f]|enter|cancel|clear|backspace|help|decPoint|shift|doubleZero|tripleZero)$|^fdk(0[1
-9]|[12][0-9]|3[0-2])$|.+
```

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

10.4.2 Keyboard.EnterDataEvent

This mandatory event notifies the application when the device is ready for the user to start entering data.

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

10.4.3 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
{		
"data": [{	array (object)	
"xPos": 0,	integer	Yes
"yPos": 0,	integer	Yes
"xSize": 0,	integer	Yes
"ySize": 0,	integer	Yes
"float": {	object	
"x": false,	boolean	
"y": false	boolean	
},		
"keys": [{	array (object)	
"xPos": 0,	integer	Yes
"yPos": 0,	integer	Yes
"xSize": 1,	integer	Yes
"ySize": 1,	integer	Yes
"key": "one",	string	
"shiftKey": "a"	string	
}]		
}],		
"pin": [{	array (object)	
"xPos": 0,	integer	Yes
"yPos": 0,	integer	Yes
"xSize": 0,	integer	Yes
"ySize": 0,	integer	Yes
"float": {	object	
"x": false,	boolean	
"y": false	boolean	
},		
"keys": [{	array (object)	
"xPos": 0,	integer	Yes
"yPos": 0,	integer	Yes
"xSize": 1,	integer	Yes
"ySize": 1,	integer	Yes
"key": "one",	string	

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

Payload	Type	Required
"shiftKey": "a"	string	
}]		
}],		
"secure": [{	array (object)	
"xPos": 0,	integer	Yes
"yPos": 0,	integer	Yes
"xSize": 0,	integer	Yes
"ySize": 0,	integer	Yes
"float": {	object	
"x": false,	boolean	
"y": false	boolean	
},		
"keys": [{	array (object)	
"xPos": 0,	integer	Yes
"yPos": 0,	integer	Yes
"xSize": 1,	integer	Yes
"ySize": 1,	integer	Yes
"key": "one",	string	
"shiftKey": "a"	string	
}]		
}]		
}		

Properties

There can be one or more frames included. A Physical Frame can only contain Physical Keys. It can contain Physical Keys positioned on the edge of the screen (for example, FDKs) or Physical Keys not positioned on the edge of the screen (for example EPP) but cannot contain both. A ETS can only contain Touch Keys. To determine the frame type, frameXSize and frameYSize should be checked. Refer to the layout table for the different types of frames, and see the diagram for an example.

data
The layout for the DataEntry command.

data/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

data/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

data/xSize
For ETS, specifies the width of the frame. For all other device types, this value is ignored

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

Properties
data/ySize For ETS, specifies the height of the frame. For all other device types, this value is ignored
data/float Specifies if the device can float the touch keyboards
data/float/x Specifies that the device will randomly shift the layout in a horizontal direction default: false
data/float/y Specifies that the device will randomly shift the layout in a vertical direction default: false
data/keys Defining details of the keys in the keyboard.

data/keys/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.
Property value constraints:
`minimum: 0`
`maximum: 999`

data/keys/yPos
Specifies the position of the top left corner of the Function Key (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.
Property value constraints:
`minimum: 0`
`maximum: 999`

data/keys/xSize
Specifies the Function Key (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.
Property value constraints:
`minimum: 1`
`maximum: 1000`

data/keys/ySize
Specifies the Function Key (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.
Property value constraints:
`minimum: 1`
`maximum: 1000`

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

Properties
data/keys/key Specifies the Function Key associated with the physical area in non-shifted mode. This property is not required if the device doesn't support. Property value constraints: `pattern: ^(one|two|three|four|five|six|seven|eight|nine|[a-f]|enter|cancel|clear|backspace|help|decPoint|shift|doubleZero|tripleZero)$\|^fdk(0[1-9]\|[12][0-9]\|3[0-2])$|.+`
data/keys/shiftKey Specifies the Function Key associated with the physical key in shifted mode. This property is not required if the device doesn't support. Property value constraints: `pattern: ^(one|two|three|four|five|six|seven|eight|nine|[a-f]|enter|cancel|clear|backspace|help|decPoint|shift|doubleZero|tripleZero)$\|^fdk(0[1-9]\|[12][0-9]\|3[0-2])$|.+`
pin The layout for the PinEntry command.
pin/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
pin/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
pin/xSize For ETS, specifies the width of the frame. For all other device types, this value is ignored
pin/ySize For ETS, specifies the height of the frame. For all other device types, this value is ignored
pin/float Specifies if the device can float the touch keyboards
pin/float/x Specifies that the device will randomly shift the layout in a horizontal direction default: false
pin/float/y Specifies that the device will randomly shift the layout in a vertical direction default: false
pin/keys Defining details of the keys in the keyboard.

pin/keys/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.
Property value constraints:
`minimum: 0`
`maximum: 999`

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

Properties

pin/keys/yPos
Specifies the position of the top left corner of the Function Key (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.
Property value constraints:
`minimum: 0`
`maximum: 999`

pin/keys/xSize
Specifies the Function Key (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.
Property value constraints:
`minimum: 1`
`maximum: 1000`

pin/keys/ySize
Specifies the Function Key (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.
Property value constraints:
`minimum: 1`
`maximum: 1000`

pin/keys/key Specifies the Function Key associated with the physical area in non-shifted mode. This property is not required if the device doesn't support. Property value constraints: `pattern: ^(one|two|three|four|five|six|seven|eight|nine|[a-f]|enter|cancel|clear|backspace|help|decPoint|shift|doubleZero|tripleZero)$\|^fdk(0[1-9]\|[12][0-9]\|3[0-2])$|.+`
pin/keys/shiftKey Specifies the Function Key associated with the physical key in shifted mode. This property is not required if the device doesn't support. Property value constraints: `pattern: ^(one|two|three|four|five|six|seven|eight|nine|[a-f]|enter|cancel|clear|backspace|help|decPoint|shift|doubleZero|tripleZero)$\|^fdk(0[1-9]\|[12][0-9]\|3[0-2])$|.+`
secure The layout for the SecureKeyEntry command.
secure/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
secure/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
secure/xSize For ETS, specifies the width of the frame. For all other device types, this value is ignored
secure/ySize For ETS, specifies the height of the frame. For all other device types, this value is ignored

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

Properties
secure/float Specifies if the device can float the touch keyboards
secure/float/x Specifies that the device will randomly shift the layout in a horizontal direction default: false
secure/float/y Specifies that the device will randomly shift the layout in a vertical direction default: false
secure/keys Defining details of the keys in the keyboard.

secure/keys/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.
Property value constraints:
`minimum: 0`
`maximum: 999`

secure/keys/yPos
Specifies the position of the top left corner of the Function Key (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.
Property value constraints:
`minimum: 0`
`maximum: 999`

secure/keys/xSize
Specifies the Function Key (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.
Property value constraints:
`minimum: 1`
`maximum: 1000`

secure/keys/ySize
Specifies the Function Key (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.
Property value constraints:
`minimum: 1`
`maximum: 1000`

secure/keys/key Specifies the Function Key associated with the physical area in non-shifted mode. This property is not required if the device doesn't support. Property value constraints: `pattern: ^(one|two|three|four|five|six|seven|eight|nine|[a-f]|enter|cancel|clear|backspace|help|decPoint|shift|doubleZero|tripleZero)$\|^fdk(0[1-9]\|[12][0-9]\|3[0-2])$|.+`

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

Properties
secure/keys/shiftKey Specifies the Function Key associated with the physical key in shifted mode. This property is not required if the device doesn't support. Property value constraints: `pattern: ^(one|two|three|four|five|six|seven|eight|nine|[a-f]|enter|cancel|clear|backspace|help|decPoint|shift|doubleZero|tripleZero)$\|^fdk(0[1-9]\|[12][0-9]\|3[0-2])$|.+`

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

11. PinPad Interface

This chapter defines the PinPad interface functionality and messages.

11.1 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

11.2 General Information

11.2.1 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.

11.2.2 Default Keyname

The DUKPT IPEK key is given a fixed name so multi-vendor applications 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).

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

11.3 Command Messages

11.3.1 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 canceled.
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	
`"applicationIdentifier": Add example to YAML`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

manufacturerIdentifier
Returns the manufacturer identifier of the PIN device. This value is not set if the manufacturer identifier is not available. This property is distinct from the HSM key pair that may be reported in the extra property by the Capabilities command.

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

Properties
modelIdentifier Returns the model identifier of the PIN device. This value is not set if the model identifier is not available.
hardwareIdentifier Returns the hardware identifier of the PIN device. This value is not set if the hardware identifier is not available.
firmwareIdentifier Returns the firmware identifier of the PIN device. This value is not set if the firmware identifier is not available.
applicationIdentifier Returns the application identifier of the PIN device. This value is not set if the application identifier is not available.

Event Messages

None

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

11.3.2 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 application. This command will clear the PIN unless the application has requested that the PIN be maintained through the PinPad.MaintinPin command.

Command Message

Payload	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"validationData": Add example to YAML,`	string	
`"offset": Add example to YAML,`	string	
`"padding": 00,`	undefined	
`"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 canceled.

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.

Property value constraints:

```
pattern: ^[0-9]{16}$
```

offset

ASCII string defining the offset data for the PIN block as an ASCII string. if this property is omitted then no offset is used. The character must be in the ranges '0' to '9', 'a' to 'f' and 'A' to 'F'.

Property value constraints:

```
pattern: ^[0-9a-fA-F]{1,16}$
```

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 character is in the range 30 to 39 ('0' to ⁄9⁄), 41 to 46 (⁄a⁄ to ⁄f⁄), or 61 to 66 (⁄A⁄ to ⁄F⁄), padding is applied before the validation data is compressed.

Property value constraints:

```
pattern: ^0[0-9a-fA-F]$|^3[0-9]$|^4[1-6]$|^6[1-6]$
```

default: 00

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

Properties
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*.
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 keyUsage 'V0' attribute.
keyEncKey If this property is omitted, 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). Property value constraints: `pattern: ^[0-9]{16}$`

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "keyNotFound",`	string	
`"result": false`	boolean	
`}`		

Properties
completionCode The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

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

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 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

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

11.3.3 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 application. This command will clear the PIN unless the application has requested that the PIN be maintained through the PinPad.MaintinPin command.

Command Message

Payload	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"pan": Add example to YAML,`	string	
`"pvv": Add example to YAML,`	string	
`"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 canceled.
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.
Property value constraints:
`pattern: ^[0-9]{23}$`

pvv
PIN Validation Value from track data.
Property value constraints:
`pattern: ^[0-9]{4,}$`

key
Name of the validation key. The key referenced by key must have the keyUsage 'V2' attribute.

keyEncKey
If this property is omitted, 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	

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

Payload	Type	Required
`"result": false`	boolean	
`}`		

Properties

completionCode
The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

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

11.3.4 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 application. This command will clear the PIN unless the application has requested that the PIN be maintained through the PinPad.MaintinPin command.

Command Message

Payload	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"presentAlgorithm": "presentClear",`	string	
`"chipProtocol": Add example to YAML,`	string	
`"chipData": Add example to YAML,`	string	
`"algorithmData": {`	object	
`"pinPointer": 0,`	integer	Yes
`"pinOffset": 0`	integer	Yes
`}`		
`}`		

Properties
timeout Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be canceled. default: 0
presentAlgorithm Specifies the algorithm that is used for presentation. Possible values are: (see presentationAlgorithms).
chipProtocol Identifies the protocol that is used to communicate with the chip. Possible values are: (see command chipProtocols in the Identification Card Device Class Interface)
chipData The Base64 encoded data to be sent to the chip.
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* property where to start inserting the PIN. The leftmost bit numbered zero.

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

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
The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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 property contains the same value as the corresponding property in the input.

chipData
The Base64 encoded data responded from the chip.

Event Messages

None

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

11.3.5 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 canceled.
default: 0

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

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

11.3.6 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 canceled.
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

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

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

11.3.7 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 unsupportedCommand 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 commad 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
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"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	
`"cryptoMethod": "ecb"`	string	
`}`		
`}`		

Properties

completionCode

The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription

If included, this contains additional vendor dependent information to assist with problem resolution.

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.

© 2021 CEN 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 canceled. 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, this property can be omitted. Used for DIEBOLD 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). Property value constraints: `pattern: ^[0-9a-fA-F]{2,}$`
xorData If the formatted PIN is encrypted twice to build the resulting PIN block, this data can be used to modify the result of the first encryption by an XOR-operation. This parameter is a string of hexadecimal data that must be converted by the application, e.g. 0x0123456789ABCDEF must be converted to 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x41 0x42 0x43 0x44 0x45 0x46 and terminated with 0x00. In other words the application would set xorData to “0123456789ABCDEF”. The hex digits 0xA to 0xF can be represented by characters in the ranges ‘a’ to ‘f’ or ‘A’ to ‘F’. If this value is omitted no XOR-operation will be performed. If the formatted PIN is not encrypted twice (i.e. if the secondEncKey property is omitted) this parameter is ignored. Property value constraints: `pattern: ^[0-9a-fA-F]{2,}$`

padding
Specifies the padding character. The valid range is 0 to 15. Only the least significant nibble is used. This property is ignored for PIN block formats with fixed, sequential or random padding.
Property value constraints:
`minimum: 0`
`maximum: 15`

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

Properties
format Specifies the format of the PIN block. Possible values are: (see 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." • `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 property 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 property can be omitted if no second encryption required. The key referenced by *secondEncKey* must have the keyUsage 'P0' 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 pinBlockAttributes. The values specified must be compatible with the key identified by *key*.

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

Properties
pinBlockAttributes/cryptoMethod This parameter specifies the cryptographic method cryptomethod that will be used with the encryption algorithm. If the algorithm is 'A', 'D', or 'T', then this property 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 the algorithm is 'R', then this property 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 The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

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

11.3.8 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 pinBlockAttributes capability. This command will clear the PIN unless the application has requested that the PIN be maintained through the PinPad.MaintinPin command.

Command Message

Payload	Type	Required
`{`		
`"customerData": Add example to YAML,`	string	Yes
`"xorData": Add example to YAML,`	string	Yes
`"padding": 0,`	integer	Yes
`"format": "ibm3624",`	string	Yes
`"key": Add example to YAML,`	string	
`"secondEncKey": Add example to YAML,`	string	
`"pinBlockAttributes": {`	object	Yes
`"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, this property can be omitted. Used for DIEBOLD 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).

Property value constraints:

```
pattern: ^[0-9a-fA-F]{2,}$
```

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 application, 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 application 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 omitted no XOR-operation will be performed. If the formatted PIN is not encrypted twice (i.e. if the secondEncKey property is omitted) this parameter is ignored.

Property value constraints:

```
pattern: ^[0-9a-fA-F]{2,}$
```

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

Properties

padding

Specifies the padding character. The valid range is 0 to 15. Only the least significant nibble is used. This property is ignored for PIN block formats with fixed, sequential or random padding.

Property value constraints:

```
minimum: 0
maximum: 15
```

format

Specifies the format of the PIN block. Possible values are: (see 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."
- `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 property 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 property can be omitted if no second encryption required. The key referenced by *secondEncKey* must have the keyUsage 'P0' attribute. If this specifies a double-length or triple-length key, triple DES encryption will be performed.

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

Properties
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 pinBlockAttributes. The values specified must be compatible with the key identified by *key*.
pinBlockAttributes/cryptoMethod This parameter specifies the cryptographic method cryptomethod that will be used with the encryption algorithm. If the algorithm is 'A', 'D', or 'T', then this property 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 the algorithm is 'R', then this property 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 The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

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

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 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 Base64 encoded encrypted PIN block.

Event Messages

- PinPad.DUKPTKSNEvent

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

11.4 Event Messages

11.4.1 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	Yes
`"ksn": Add example to YAML`	string	Yes
`}`		

Properties
key Specifies the name of the DUKPT Key derivation key.
ksn Base64 encoded structure that contains the KSN.

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

11.5 Unsolicited Messages

11.5.1 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 the errorCode property.

Payload	Type	Required
`{`		
`"keyName": Add example to YAML,`	string	Yes
`"errorCode": "keyNotFound"`	string	Yes
`}`		

Properties

keyName
Specifies the name of the key that caused the error.

errorCode
Specifies the type of illegal key access that occurred. The following values are possible:

- `keyNotFound` - The specified key was not loaded or attempting to delete a non-existent key.
- `keyNoValue` - The specified key is not loaded.
- `useViolation` - The specified use is not supported by this key.
- `algorithmNotSupp` - The specified algorithm is not supported by this key.

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

12. Printer Interface

This chapter defines the Printer interface functionality and messages.

12.1 Summary

This specification describes the functionality of the services provided by banking printers and scanning devices under XFS, focusing on the following areas:

- application 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.

12.2 General Information

12.2.1 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 application 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 application. 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 application.
- **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

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

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.

12.2.2 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 application 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 application 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 application 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 application authoring data. The figure below outlines the printing process from the application'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 application 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 application. 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.

12.2.3 Command Overview

The basic operation of the print devices is managed using the two primary commands:

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

- **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 an application to assemble the required set of fields for a form before locking the service. This minimizes the time that each application 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 applications.

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).

12.2.4 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.

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.

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

- 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 (0xFFFE) 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.
- 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.

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)

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

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.

Form Definition

Attributes are not required in any mandatory order within a Form definition.

XFSFORM			
XFSFORM		*formname*	
BEGIN			
(required)	**UNIT**	*base*,	Base resolution unit for form definition: MM INCH ROWCOLUMN
		x,	Horizontal base unit fraction
		y	Vertical base unit fraction
(required)	**SIZE**	*width*,	Width of form
		height	Height of form
(required)	**ALIGNMENT**	*alignment*,	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.
		xoffset,	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)
		yoffset	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	*type*	Orientation of form: PORTRAIT (default) LANDSCAPE
	SKEW	*skewfactor*	Maximum skew factor in degrees (default = 0)
	VERSION	*major*,	Major version number
		minor,	Minor version number
		date,	Creation/modification date
		author	Author of the form
	CPI	*cpi*	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.

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

XFSFORM			
	LPI	*lpi*	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	*pointsize*	This value specifies the default POINTSIZE within the form.
	COPYRIGHT	*copyright*	Copyright entry
	TITLE	*title*	Title of form
	COMMENT	*comment*	Comment section
	USERPROMPT	*prompt*	Prompt string for user interaction
	[XFSFIELD BEGIN ... END]	*fieldname*	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]	*framename*	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]	*subformname*	One subform definition (as defined in the next section) for each subform in the form. The subformname within a form must be unique.
END			

SubForm Definition

XFSSUBFORM			
XFSSUBFORM		*subformname*	
BEGIN			
(required)	**POSITION**	*x,*	Horizontal position (relative to left side of form)
		y or *(y, z)*	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**	*width,*	Width of subform. Width must not exceed width of form.
		height	Height of subform. Height must not exceed height of form.
	[XFSFIELD BEGIN ... END]	*fieldname*	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.

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

XFSSUBFORM			
	[XFSFRAME BEGIN ... END]	*framename*	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.
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
```

Field Definition

XFSFIELD			
XFSFIELD		*fieldname*	The fieldname within a form and its optional subforms must be unique.
BEGIN			
(required)	**POSITION**	*x,*	Horizontal position (relative to left side of form/subform).
		y or *(y, z)*	Vertical position (relative to top of form/subform). Format (y, z) 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 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 relative to top of the 1st form/subform.
	FOLLOWS	*fieldname*	Print this field directly following the field with the name *fieldname*; positioning information is ignored. See the description of Printer.PrintForm. If FOLLOWS is omitted, then fields are printed in the order that they appear in the form definition.

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

XFSFIELD			
	HEADER	*N*	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.
		N-N	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.
		ALL	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	*N*	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.
		N-N	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.
		ALL	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	*N*	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.
		N-N	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.
		ALL	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	*side*	Side of form where field is positioned: FRONT (default) BACK
(required)	**SIZE**	*width*,	Field width.
		height	Field height.
	INDEX	*repeatcount*	Count how often this field is repeated in the form, INDEX fields are fixed length (default is no INDEX field).
		xoffset,	Horizontal offset for next field.
		yoffset	Vertical offset for next field.

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

XFSFIELD			
	TYPE	*fieldtype*	Type of field: TEXT (default) MICR OCR MSF BARCODE GRAPHIC PAGEMARK
	SCALING	*scalingtype*	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	*hriposition*	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	*coercivity*	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	*class*	Field class: OPTIONAL (default) STATIC REQUIRED
	ACCESS	*access*	Access rights of field: WRITE (default) READ READWRITE
	OVERFLOW	*overflow*	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

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

XFSFIELD			
	STYLE	*style*	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	*case*	Convert field contents to: NOCHANGE (default) UPPER LOWER
	HORIZONTAL	*justify*	Horizontal alignment of field contents: LEFT (default) RIGHT CENTER JUSTIFY
	VERTICAL	*justify*	Vertical alignment of field contents: BOTTOM (default) CENTER TOP
	COLOR	*color*	Color name: BLACK (default) WHITE GRAY RED BLUE GREEN YELLOW

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

XFSFIELD			
	RGBCOLOR	*R, G, B*	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.
	FONT	*fontname*	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	*pointsize*	Point size. If unspecified, the point size defaults to the POINTSIZE defined for the form.
	CPI	*cpi*	Characters per inch. If unspecified, the CPI defaults to the CPI defined for the form.
	LPI	*lpi*	Lines per inch. If unspecified, the LPI defaults to the LPI defined for the form.
	FORMAT	*formatstring*	This is an application defined input field describing how the application should format the data. This may be interpreted by the Service Provider.
	INITIALVALUE	*value*	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.

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

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.

Frame Definition

XFSFRAME			
XFSGRAME		*framename*	
BEGIN			
(required)	**POSITION**	*X,*	Horizontal position of top left corner of the frame (relative to left side of form/subform).

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

XFSFRAME			
		Y or *(Y, Z)*	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	*fieldname*	Frames the field with the name *fieldname*, 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	*N*	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.
		N-N	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.
		ALL	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	*N*	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.
		N-N	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.
		ALL	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	*side*	Side of form where this frame is positioned: FRONT (default) BACK
(required)	**SIZE**	*width*,	Frame width in base horizontal units for the form.
		height	Frame height in base vertical units for the form.
	REPEATONX	*repeatcount*,	Count how often this frame is repeated horizontally in the form.
		xoffset	Horizontal offset for next frame in base horizontal units.

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

XFSFRAME			
	REPEATONY	*repeatcount*,	Count how often this frame is repeated vertically in the form.
		yoffset	Vertical offset for next frame in base vertical units.
	TYPE	*frametype*	Type of frame: RECTANGLE (default) ROUNDED_CORNER ELLIPSE
	CLASS	*class*	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	*overflow*	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	*style*	Frame line attributes: SINGLE_THIN (default) DOUBLE_THIN SINGLE_THICK DOUBLE_THICK DOTTED
	COLOR	*color*	Color name for frame lines: BLACK (default) WHITE GRAY RED BLUE GREEN YELLOW
	RGBCOLOR	*R, G, B*	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	*color*	Color name for interior of frame: BLACK WHITE (default) GRAY RED BLUE GREEN YELLOW

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

XFSFRAME			
	RGBFILLCOLOR	*R, G, B*	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	*style*	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	*substitute sign*	Character that is used as substitute sign when a character in a read field cannot be read
	TITLE	*fieldname*	Uses the field with the name as the title of the frame. Positioning information of the field is ignored.
	HORIZONTAL	*justify*	Horizontal alignment of the frame title: LEFT (default) CENTER RIGHT
	VERTICAL	*justify*	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 it bottom right corner is one base unit below and to the right to 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.

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

The **SIDE** attributes of the **XFSFRAME** and the **XFSFIELD**s 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.

Sample 1 - Simple Framing

The form:

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

```
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
```

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

```
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
0123456789123001	$17465.12
0123456789123002	$2458.23
0123456789123003	$6542.78

Sample 2 - Framing With Title

The form:

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

```
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:

Sample 3 - Framing With Filled Interior

The form:

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

```
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:

Sample 4 - Repeated Framing

The form:

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

```
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

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.

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

XFSMEDIA			
XFSMEDIA		*medianame*	
BEGIN			
	TYPE	*type*	Predefined media types are: GENERIC (default) MULTIPART PASSBOOK
	SOURCE	*source*	Paper source: ANY (default) UPPER LOWER EXTERNAL (envelope tray or single sheet feed tray) AUX AUX2 PARK
(required)	**UNIT**	*base*,	Base resolution unit for form definition: MM INCH ROWCOLUMN
		x,	Horizontal base unit fraction
		y	Vertical base unit fraction
(required)	**SIZE**	*width*,	Width of physical media
		height	Height of physical media (0 = unlimited, i.e. roll paper)
	PRINTAREA	*x*,	Printable area relative to top left corner of physical media (default = physical size of media)
		y,	
		width,	
		height	
	RESTRICTED	*x*,	Restricted area relative to top left corner of physical media (default = no restricted area)
		y	

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

XFSMEDIA			
		width,	
		height	
	FOLD	*fold*,	Type of passbook: HORIZONTAL (default) VERTICAL
	STAGGERING	*staggering*	Staggering of passbook from top (default = 0)
	PAGE	*count*	Number of pages in passbook (default = 0)
	LINES	*count*	Number of printable lines (default = 0)
END			

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 applications 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.

12.2.5 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.

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.

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

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*.

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

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.

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

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.

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

- 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.

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

12.3 Command Messages

12.3.1 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 canceled.
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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

formList
The list of form names.

Event Messages

None

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

12.3.2 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 canceled.
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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

mediaList
The list of media names.

Event Messages

None

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

12.3.3 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 canceled.
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)	
`}`		

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

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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 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.

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

Properties
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

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

12.3.4 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

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

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

Payload	Type	Required
`"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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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:

- `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.

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

Properties
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.

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

Properties
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

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

12.3.5 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
`{`		
`"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 canceled. 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	
`"exampleProperty1": {`	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	
`},`		
`"exampleProperty2": {`	object	

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

Payload	Type	Required
`See exampleProperty1 properties.`		
`}`		
`}`		
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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/exampleProperty1 (example name)

fields/exampleProperty1/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/exampleProperty1/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/exampleProperty1/class
Specifies the class of the field as one of the following:

- `static` - The field data cannot be set by the application.
- `optional` - The field data can be set by the application.
- `required` - The field data must be set by the application.

fields/exampleProperty1/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.

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

Properties
fields/exampleProperty1/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/exampleProperty1/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/exampleProperty1/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/exampleProperty1/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

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

12.3.6 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 canceled.
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	
`"codelineFormat": "cmc7",`	string	
`"charMapping": Add example to YAML`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

codelineFormat
Specifies the code-line format that is being reported as one of the following:

- `cmc7` - CMC7 mapping.
- `e13b` - E13B mapping.

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

Properties

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
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

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

12.3.7 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 canceled.

default: 0

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

Properties
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*. An application 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 application 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.

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

Properties
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 *Pinter.PrintForm* or *Printer.PrintRawFile* commands.

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "noMediaPresent"`	string	
`}`		

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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. • `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

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

12.3.8 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": {`	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	

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

Payload	Type	Required
`"clearBuffer": false`	boolean	
`},`		
`"fields": {`	object	
`"exampleProperty1": Add example to YAML,`	string	
`"exampleProperty2": Add example to YAML`	string	
`},`		
`"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 canceled.
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.
Property value constraints:
`minimum: 0`

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.

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

Properties
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 application 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 application 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.

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

Properties
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 *Pinter.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.
fields/exampleProperty1 (example name)
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 The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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

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

- Printer.MediaRejectedEvent
- Printer.MediaAutoRetractedEvent

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

12.3.9 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 application 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	

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

Payload	Type	Required
`"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 canceled.
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 application 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.

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

Properties
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 *Pinter.PrintForm* or *Printer.PrintRawFile* commands.

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "formNotFound",`	string	
`"fields": {`	object	
`"exampleProperty1": Add example to YAML,`	string	
`"exampleProperty2": Add example to YAML`	string	
`}`		
`}`		

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

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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.
fields/exampleProperty1 (example name)

Event Messages

- Printer.NoMediaEvent
- Printer.MediaInsertedEvent
- Printer.FieldErrorEvent
- Printer.FieldWarningEvent
- Printer.RetractBinThresholdEvent
- Printer.MediaTakenEvent
- Printer.InkThresholdEvent
- Printer.LampThresholdEvent
- Printer.MediaRejectedEvent

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

12.3.10 Printer.RawData

This command is used to send raw data (a byte string of device dependent data) to the physical device.

Applications 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.
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 canceled.
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	
`}`		

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

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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

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

12.3.11 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

Payload	Type	Required
`{`		
`"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 canceled.
default: 0

base
Specifies the base unit of measurement of the media and can be one of the following values:

- `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	
`}`		

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

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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

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

12.3.12 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
`{`		
`"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 canceled.
default: 0

binNumber
Specifies the height of the media in terms of the base vertical resolution.

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

- Printer.RetractBinThresholdEvent

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

12.3.13 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	
`"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 canceled.

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.

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

Properties
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:TempFrontImage.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:TempBackImage.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.

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

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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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.

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

Properties
images/front/status Status of data source. Possible values are: • `ok` - The data is OK. • `notSupp` - The data source is not supported. • `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

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

12.3.14 Printer.Reset

This command is used by the application 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 application 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 canceled.
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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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

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

12.3.15 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 canceled.
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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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

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

12.3.16 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 canceled.
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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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

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

12.3.17 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	
}		

© 2021 CEN 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 canceled. 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 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 application 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.

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

Properties
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 *Pinter.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 The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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

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

12.3.18 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 applications 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 canceled.
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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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

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

12.3.19 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 canceled. 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.

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

Properties
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 The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

- Printer.PaperThresholdEvent
- Printer.TonerThresholdEvent
- Printer.InkThresholdEvent
- Printer.LampThresholdEvent

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

12.3.20 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 canceled.
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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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

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

12.3.21 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 canceled.
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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

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

12.4 Event Messages

12.4.1 Printer.MediaPresentedEvent

This event is used to indicate when media has been presented to the customer for removal.

Payload	Type	Required
`{`		
`"wadIndex": 1,`	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.

Property value constraints:

```
minimum: 1
```

totalWads

Specifies the total number of wads in the print job, zero if the total number of wads is not known.

Property value constraints:

```
minimum: 0
```

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

12.4.2 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 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 application 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.

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

12.4.3 Printer.MediaInsertedEvent

This event specifies that the physical media has been inserted into the device.

The application may use this event to, for example, remove a message box from the screen telling the user to insert media.

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

12.4.4 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:

- `required` - The specified field must be supplied by the application.
- `staticOverwrite` - The specified field is static and thus cannot be overwritten by the application.
- `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.

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

12.4.5 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 application.
- `staticOverwrite` - The specified field is static and thus cannot be overwritten by the application.
- `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.

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

12.4.6 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 application 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.

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

12.5 Unsolicited Messages

12.5.1 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.

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

12.5.2 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.

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

12.5.3 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": 1,`	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.

Property value constraints:

```
minimum: 1
```

totalWads

Specifies the total number of wads in the print job, zero if the total number of wads is not known.

Property value constraints:

```
minimum: 0
```

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

12.5.4 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*.

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

12.5.5 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.

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

12.5.6 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

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.

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

12.5.7 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*.

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

12.5.8 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.

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

12.5.9 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

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.

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

12.5.10 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.

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

12.5.11 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.

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

12.5.12 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.

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

13. Text Terminal Interface

This chapter defines the Text Terminal interface functionality and messages.

13.1 Summary

This section describes the functions provided by a generic Text Terminal Unit interface. 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

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.

13.2 General Information

13.2.1 References

None

13.2.2 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.
- **Coments**
  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.

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

- 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

XFS 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 applications 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.
- 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		***formname****	
BEGIN			
(required)	**SIZE**	*width* *height*	Width of form Height of form
	VERSION	*major*, *minor*, *date**, *author**	Major version number (default 0) Minor version number (default 0) Creation/modification date Author of form
(required)	**LANGUAGE**	*languageID*	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	*copyright**	Copyright entry
	TITLE	*title**	Title of form
	COMMENT	*comment**	Comment section
	[FIELD	*fieldname**	One field definition (as defined in the next section) for each field in the form
	BEGIN **. . .** **END]**		
END			

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

Field Definition

FIELD		*fieldname**	
BEGIN			
	LANGUAGE	*languageID*	Language used for this field. See Form definition for detailed description. If unspecified defaults to form definition Language specification.
(required)	**POSITION**	*x*, *y*	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**	*width*, *height*	Field width Field height
	TYPE	*fieldtype*	Type of field: **Text** (default) **Invisible** **Password** (contents is echoed with '*') **Graphic** (ignored for TextTerminal.ReadForm commands)
	SCALING	*scalingtype*	Information on how to size the Graphic within the field: **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. SCALING is only relevant for Graphics field types
	CLASS	*class*	Field class: **Optional** (default) **Static** **Required**
	KEYS	*keys*	Accepted input key types: **Numeric** **Hexadecimal** **Alphanumeric** This is an optional field where the default value is vendor dependent.
	ACCESS	*access*	Access rights of field: **Write** (default) **Read** **ReadWrite**
	OVERFLOW	*overflow*	Action on field overflow: **Terminate** (default) **Truncate** **OverWrite**
	STYLE	*style*	Display attributes as a combination of the following, ORed together using the '
	HORIZONTAL	*justify*	Horizontal alignment of field contents: **Left** (default) **Right** **Center**

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

FIELD		***fieldname****	
	FORMAT	*formatstring**	This is an application defined input field describing how the application should format the data. This may be interpreted by the Service Provider.
	INITIALVALUE	*value**	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			

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

13.3 Command Messages

13.3.1 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 canceled.
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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

formList
Array of the form names.

Event Messages

None

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

13.3.2 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 canceled.
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	
`"errorCode": "formNotFound",`	string	
`"formName": Add example to YAML,`	string	
`"width": Add example to YAML,`	string	
`"height": Add example to YAML,`	string	
`"versionMajor": Add example to YAML,`	string	
`"versionMinor": Add example to YAML,`	string	
`"fields": [Add example to YAML],`	array (string)	
`"languageId": Add example to YAML`	string	
`}`		

Properties

completionCode
The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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.
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 specifies 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.
fields Object to a list of the field names.
languageId Specifies the language identifier for the form.

Event Messages

None

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

13.3.3 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 canceled. 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	
`"errorCode": "formNotFound",`	string	
`"fields": {`	object	
`"exampleProperty1": {`	object	
`"type": "text",`	string	
`"class": "static",`	string	
`"access": {`	object	
`"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	
`},`		

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

Payload	Type	Required
`"exampleProperty2": {`	object	
`See exampleProperty1 properties.`		
`}`		
`}`		
`}`		

Properties
completionCode The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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. • `fieldNotfound` - The specified field cannot be found. • `fieldInvalid` - The specified field is invalid.
fields Details of the field(s) requested. The key is the field name and the value contains the details of the fields.
fields/exampleProperty1 (example name)
fields/exampleProperty1/type Specifies the type of field and can be one of the following: • `text` - A text field. • `invisible` - An invisible text field. • `password` - A password field, input is echoed as '*'.
fields/exampleProperty1/class Specifies the class of the field and can be one of the following: • `static` - The field data cannot be set by the application. • `optional` - The field data can be set by the application. • `required` - The field data must be set by the application.
fields/exampleProperty1/access Specifies whether the field is to be used for input, output or both.
fields/exampleProperty1/access/read The Field is used for input from the physical device.
fields/exampleProperty1/access/write The Field is used for output to the physical device.
fields/exampleProperty1/overflow Specifies how an overflow of field data should be handle and can be one of the following: • `terminate` - Return an error and terminate display of the form. • `truncate` - Truncate the field data to fit in the field. • `overwrite` - Print the field data beyond the extents of the field boundary.

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

Properties
fields/exampleProperty1/format Format string as defined in the form for this field.
fields/exampleProperty1/languageId Specifies the language identifier for the field.

Event Messages

None

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

13.3.4 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 canceled.
default: 0

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"keys": "0123456789ABCabc",`	string	
`"commandKeys": ["enter", "cancel"]`	array (string)	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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 property is omitted if no keys of this type are present on the device.

commandKeys
Supporting command keys on the Text Terminal Unit. This property can be omitted if no command keys supported.
Property value constraints:

```
uniqueItems: true
minItems: 1
pattern:
^(enter|cancel|clear|backspace|help|doubleZero|tripleZero|arrowUp|arrowDown|arrowLef
t|arrowRight)$|^fdk(0[1-9]|[12][0-9]|3[0-2])$|.+
```

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

Event Messages

None

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

13.3.5 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 canceled. 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.

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

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	string	
`}`		

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

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

13.3.6 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 canceled. default: 0
positionX Specifies the horizontal position of the area to be cleared. Property value constraints: `minimum: 0`
positionY Specifies the vertical position of the area to be cleared. Property value constraints: `minimum: 0`
width Specifies the width position of the area to be cleared. Property value constraints: `minimum: 0`
height Specifies the height position of the area to be cleared. Property value constraints: `minimum: 0`

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	string	
`}`		

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

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

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

13.3.7 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 canceled.
default: 0

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

resolution/sizeX
TSpecifies the horizontal size of the display of the text terminal unit (the number of columns that can be displayed).
Property value constraints:
`minimum: 0`

resolution/sizeY
Specifies the vertical size of the display of the text terminal unit (the number of rows that can be displayed).
Property value constraints:
`minimum: 0`

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "resolutionNotSupported"`	string	
`}`		

Properties

completionCode
The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

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

Properties
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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

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

13.3.8 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": {`	object	
`"exampleProperty1": Add example to YAML,`	string	
`"exampleProperty2": 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 canceled.
default: 0

formName
Specifies the name of the form.

clearScreen
Specifies whether the screen is cleared before displaying the form (true) or not (false).

fields
Details of the field(s) to write. The key is the field name and value is file value containing all the printable characters (numeric and alphanumeric) to display on the text terminal unit key pad for this field.

fields/exampleProperty1 (example name)

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "formNotFound"`	string	
`}`		

Properties

completionCode
The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

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

Properties
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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

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

13.3.9 TextTerminal.ReadForm

This command is used to read data from input fields on the specified form.

Command Message

Payload	Type	Required
`{`		
`"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 canceled.
default: 0

formName
Specifies the 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": {`	object	
`"exampleProperty1": Add example to YAML,`	string	
`"exampleProperty2": Add example to YAML`	string	
`}`		
`}`		

Properties

completionCode
The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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 Details of the field(s) requested. The key is the field name and value is file value containing all the printable characters (numeric and alphanumeric) read from the text terminal unit key pad for this field.
fields/exampleProperty1 (example name)

Event Messages

None

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

13.3.10 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 canceled.

default: 0

mode

Specifies whether the position of the output is absolute or relative to the current cursor position. Possible values are:

- `relative` - The cursor is positioned relative to the current cursor position.
- `absolute` - The cursor is positioned absolute at the position specified in *posX* and *posY*.

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.

Property value constraints:

```
minimum: 0
```

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.

Property value constraints:

```
minimum: 0
```

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

Properties
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 unerlined.
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 The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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

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

13.3.11 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": "12AaBb",`	string	
`"activeCommandKeys": {`	object	
`"enter": {`	object	
`"terminate": false`	boolean	
`},`		
`"backspace": {`	object	
`See enter properties.`		
`},`		
`"oem1": {`	object	
`See enter properties.`		
`},`		
`"oem10": {`	object	
`See oem1 properties.`		
`}`		
`}`		

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

Payload	Type	Required
`}`		

Properties

timeout

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

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 'enter', 'fdk01' will not be counted.

Property value constraints:

```
minimum: 0
```

mode

Specifies whether the position of the output is absolute or relative to the current cursor position. Possible values are:

- `relative` - The cursor is positioned relative to the current cursor position.
- `absolute` - The cursor is positioned absolute at the position specified in *posX* and *posY*.

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.

Property value constraints:

```
minimum: 0
```

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.

Property value constraints:

```
minimum: 0
```

echoMode

Specifies how the user input is echoed to the screen as one of the following flags:

- `text` - The user input is echoed to the screen.
- `invisible` - The user input is not echoed to the screen.
- `password` - The keys entered by the user are echoed as the replace character on 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).

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

Properties
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 property can be omitted if no keys of this type are active keys.
activeCommandKeys Array specifying the command keys which are active during the execution of the command. The following standard names are defined: • `enter` - Enter • `cancel` - Cancel • `clear` - Clear • `backspace` - Backspace • `help` - Help • `doubleZero` - 00 • `tripleZero` - 000 • `arrowUp` - up arrow • `arrowDown` - down arrow • `arrowLeft` - left arrow • `arrowRight` - right arrow • `fdk[01-32]` - 32 FDK keys Additional non standard key names are also allowed.
activeCommandKeys/enter (example name) A standard key name. Property name constraints: `pattern: ^(enter|cancel|clear|backspace|help|doubleZero|tripleZero|arrowUp|arrowDown|arrowLeft|arrowRight)$\|^fdk(0[1-9]\|[12][0-9]\|3[0-2])$`
activeCommandKeys/enter/terminate The key is a terminate key. default: false
activeCommandKeys/oem1 (example name) A non standard key name.

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

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "keyInvalid",`	string	
`"input": Add example to YAML`	string	
`}`		

Properties

completionCode
The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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

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

13.3.12 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 canceled.
default: 0

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

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

13.3.13 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": "0123456789ABCabc",`	string	
`"activeCommandKeys": {`	object	
`"enter": {`	object	
`"terminate": false`	boolean	
`},`		
`"backspace": {`	object	
`See enter properties.`		
`},`		
`"oem1": {`	object	
`See enter properties.`		
`},`		
`"oem10": {`	object	
`See oem1 properties.`		
`}`		
`}`		
`}`		

Properties

timeout

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

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.

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

Properties
activeCommandKeys Array specifying the command keys which are active during the execution of the next TextTerminal.ReadForm command. The following standard names are defined: • `enter` - Enter • `cancel` - Cancel • `clear` - Clear • `backspace` - Backspace • `help` - Help • `doubleZero` - 00 • `tripleZero` - 000 • `arrowUp` - up arrow • `arrowDown` - down arrow • `arrowLeft` - left arrow • `arrowRight` - right arrow • `fdk[01-32]` - 32 FDK keys Additional non standard key names are also allowed.
activeCommandKeys/enter (example name) A standard key name. Property name constraints: `pattern: ^(enter|cancel|clear|backspace|help|doubleZero|tripleZero|arrowUp|arrowDown|arrowLeft|arrowRight)$\|^fdk(0[1-9]\|[12][0-9]\|3[0-2])$`
activeCommandKeys/enter/terminate The key is a terminate key. default: false
activeCommandKeys/oem1 (example name) A non standard key name.

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "keyInvalid"`	string	
`}`		

Properties
completionCode The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

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

Properties
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

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

13.4 Unsolicited Messages

13.4.1 TextTerminal.FieldErrorEvent

This event specifies that a fatal error has occurred while processing a field.

Payload	Type	Required
`{`		
`"formName": Add example to YAML,`	string	Yes
`"fieldName": Add example to YAML,`	string	Yes
`"failure": "required"`	string	Yes
`}`		

Properties

formName
Specifies the form name.

fieldName
Specifies the field name.

failure
Specifies the type of failure and can be one of the following:

- `required` - The specified field must be supplied by the application.
- `staticOvwr` - The specified field is static and thus cannot be overwritten by the application.
- `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.
- `typeNotSupported` - The form field type is not supported with device.
- `charSetForm` - Service Provider does not support character set specified in form.

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

13.4.2 TextTerminal.FieldWarningEvent

This event is used to specify that a non-fatal error has occurred while processing a field.

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

13.4.3 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 application more details about individual key presses this information may also be used if the device has no internal display unit and the application has to manage the display of the entered digits.

Payload	Type	Required
`{`		
`"key": "0",`	string	Yes
`"commandKey": "enter"`	string	Yes
`}`		

Properties

key

On a numeric or alphanumeric key press this parameter holds the value of the key pressed. This property is not required 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. 'enter'. This property is not required when no command key was pressed.

Property value constraints:

```
pattern:
^(enter|cancel|clear|backspace|help|doubleZero|tripleZero|arrowUp|arrowDown|arrowLef
t|arrowRight)$|^fdk(0[1-9]|[12][0-9]|3[0-2])$|.+
```

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

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 application data and transmits it to the host system.

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

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
{		
"timeout": 5000,	integer	
"symbologies": {	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	

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

Payload	Type	Required
"msi": false,	boolean	
"plessey": false,	boolean	
"std2Of5": false,	boolean	
"std2Of5Iata": 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	
}		
}		

© 2021 CEN 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 canceled. default: 0
symbologies Specifies the sub-set of bar code symbologies that the application 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 1and 5 digit add-on
symbologies/upcA_2 UPC-A with2 digit add-on

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

Properties
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

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

Properties
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

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

Properties
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 The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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 application is responsible for checking the completeness and validity of the data.
readOutput/symbologyName A vendor dependent symbology identifier for the symbology recognized.

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

Event Messages

None

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

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

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 canceled.
default: 0

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

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

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 interface is used.
- Reading / writing smart cards - CardReader service, however when combined writing smart card / embossing is performed the CardEmbosser interface 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.

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 ("); 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.

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

- 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.

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:

- 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)

Embossing Form Definition

Attributes are not required in any mandatory order within a Form definition.

XFSFORM			
XFSFORM		*formname*	
BEGIN			
(required)	**UNIT**	*base*,	Base resolution unit for form definition: MM INCH ROWCOLUMN
		x,	Horizontal base unit fraction
		y	Vertical base unit fraction
(required)	**SIZE**	*width*,	Width of form
		height	Height of form

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

XFSFORM			
(required)	**ALIGNMENT**	*alignment*,	Alignment of the form on the physical media TOPLEFT (default) TOPRIGHT BOTTOMLEFT BOTTOMRIGHT
		xoffset,	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)
		yoffset	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	*major*,	Major version number
		minor,	Minor version number
		date,	Creation/modification date
		author	Author of form
	COPYRIGHT	*copyright*	Copyright entry
	TITLE	*title*	Title of form
	COMMENT	*comment*	Comment section
	USERPROMPT	*prompt*	Prompt string for user interaction
	[XFSFIELD BEGIN ... **END]**	*fieldname*	One field definition (as defined in the next section) for each field in the form.
END			

Embossing Field Definition

XFSFIELD			
XFSFIELD		*fieldname*	
BEGIN			
(required)	**POSITION**	*x*,	Horizontal position (relative to left or right side of form, depending upon HPOSITION keyword)
		y	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)

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

XFSFIELD			
	SIDE	*side*	Side of card: FRONT (default) BACK
(required)	**SIZE**	*width*,	Field width
		height	Field height
	TYPE	*fieldtype*	Type of field: TEXT (default) OCR
	CLASS	*class*	Field class: OPTIONAL (default) STATIC REQUIRED
	CASE	*case*	Convert field contents to: NOCHANGE (default) UPPER LOWER
	HORIZONTAL	*justify*	Horizontal alignment of field contents: LEFT (default) RIGHT CENTER JUSTIFY
	VERTICAL	*justify*	Vertical alignment of field contents: BOTTOM (default) CENTER TOP
	FONT	*fontname*	Font name; in some cases this predefines the following parameters:
	POINTSIZE	*pointsize*	Point size. If unspecified, the point size defaults to the POINTSIZE defined for the form.
	CPI	*cpi*	Characters per inch
	LPI	*lpi*	Lines per inch
	FORMAT	*formatstring*	This is an application defined input field describing how the application should format the data. This may be interpreted by the Service Provider.
	INITIALVALUE	*value*	Initial value
END			

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.

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

XFSMEDIA			
XFSMEDIA		*medianame*	
BEGIN			
	TYPE	*type*	Predefined media types are: EMBOSSCARD PRINTCARD
(required)	**UNIT**	*base*,	Base resolution unit for form definition: MM INCH ROWCOLUMN
		x,	Horizontal base unit fraction
		y	Vertical base unit fraction
(required)	**SIZE**	*width*,	Width of physical media
		height	Height of physical media
	EMBOSSAREA	*x*,	Embossing or Printing area relative to top left corner of physical media (default = physical size of media)
		y,	
		width,	
		height	
	RESTRICTED	*x*,	Restricted area relative to top left corner of physical media (default = no restricted area)
		y	
		width,	
		height	
END			

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

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 canceled.
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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

formList
The list of form names.

Event Messages

None

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

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
Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be canceled.
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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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
The form name.

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

Properties
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

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

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 canceled.
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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

mediaList
The list of media names.

Event Messages

None

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

15.3.4 CardEmbosser.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
timeout Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be canceled. 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	
`}`		

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
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

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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 canceled. 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	
`"exampleProperty1": {`	object	
`"type": "text",`	string	
`"class": "static",`	string	
`"initialValue": Add example to YAML,`	string	
`"format": Add example to YAML`	string	
`},`		
`"exampleProperty2": {`	object	
`See exampleProperty1 properties.`		
`}`		
`}`		
`}`		

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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/exampleProperty1 (example name)
fields/exampleProperty1/type Specifies the type of field as one of the following: • `text` - A text field. • `ocr` - An Optical Character Recognition (OCR) field.
fields/exampleProperty1/class Specifies the class of the field as one of the following: • `static` - The field data cannot be set by the application. • `optional` - The field data can be set by the application. • `required` - The field data must be set by the application.
fields/exampleProperty1/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/exampleProperty1/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

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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.

Applications 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	
`"exampleProperty1": Add example to YAML,`	string	
`"exampleProperty2": Add example to YAML`	string	
`},`		
`"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 canceled.
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.

fields/exampleProperty1 (example name)

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
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 interface.
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 The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
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

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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 an application 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 canceled.
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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

- CardEmbosser.OutputBinThresholdEvent
- CardEmbosser.RetainBinThresholdEvent
- CardEmbosser.MediaDetectedEvent

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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 canceled.
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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Event Messages

- CardEmbosser.InputBinThresholdEvent
- CardEmbosser.TonerThresholdEvent

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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
`{`		
`"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.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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 application.
- `staticOverwrite` - The specified field is static and thus cannot be overwritten by the application.
- `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.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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 application.
- `staticOverwrite` - The specified field is static and thus cannot be overwritten by the application.
- `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.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

15.5.4 CardEmbosser.MediaRemovedEvent

This event is generated when a card is removed before completion of a write operation.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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 References

ID	Description
biometric-2	ANSI INCITS 381-2004 Information Technology - Finger Image-Based Data Interchange Format.
biometric-3	ANSI INCITS 378-2004 Information Technology - Finger Minutiae Format for Data Interchange.
biometric-4	ISO/IEC 19794-4:2005 Information technology - Biometric data interchange formats - Part 4: Finger image data.
biometric-5	ISO/IEC 19794-2:2005 Information technology - Biometric data interchange formats - Part 2: Finger minutiae data.

16.2.2 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.

16.2.3 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.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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.

16.2.4 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.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

16.2.5 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 KeyManagement interface commands can be used for key management. In this case the Biometric Service Provider would implement the biometric methods necessary to read and return data using the Biometric interface, while the key loading, reporting etc, the KeyManagement interface would be implemented in order to provide key management.

16.2.6 Biometric Device Command Flows

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.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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 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.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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 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.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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 canceled.
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	Yes
`    "type": {`	object	Yes
`      "format": "isoFid",`	string	Yes
`      "algorithm": "ecb",`	string	Yes
`      "keyName": Add example to YAML`	string	
`    }`		
`  }]`		
`}`		

Properties

completionCode
The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
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 A list of the biometric template data that were successfully imported.
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. The following values are possible: • `isoFid` - Raw ISO FID format [Ref. biometric-4]. • `isoFmd` - ISO FMD template format [Ref. biometric-5]. • `ansiFid` - Raw ANSI FID format [Ref. biometric-2]. • `ansiFmd` - ANSI FMD template format [Ref. biometric-3]. • `qso` - Raw QSO image format. • `wso` - WSQ image format. • `reservedRaw1` - Reserved for a vendor-defined Raw format. • `reservedTemplate1` - Reserved for a vendor-defined Template format. • `reservedRaw2` - Reserved for a vendor-defined Raw format. • `reservedTemplate2` - Reserved for a vendor-defined Template format. • `reservedRaw3` - Reserved for a vendor-defined Raw format. • `reservedTemplate3` - Reserved for a vendor-defined Template format.
storageList/type/algorithm Specifies the encryption algorithm. The following values are possible: • `ecb` - Triple DES with Electronic Code Book. • `cbc` - Triple DES with Cipher Block Chaining • `cfb` - Triple DES with Cipher Feed Back. • `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

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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 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
`{`		
`"timeout": 5000,`	integer	
`"type": [{`	array (object)	
`"format": "isoFid",`	string	Yes
`"algorithm": "ecb",`	string	Yes
`"keyName": Add example to YAML`	string	
`}],`		
`"numCaptures": 0,`	integer	
`"mode": "scan"`	string	
`}`		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be canceled.

default: 0

type

Array of data, each data element of which represents the data type(s) in which the data should be returned in the completion payload. If no data is to be returned *type* can be omitted. 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.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
type/format Specifies the format of the template data. The following values are possible: • `isoFid` - Raw ISO FID format [Ref. biometric-4]. • `isoFmd` - ISO FMD template format [Ref. biometric-5]. • `ansiFid` - Raw ANSI FID format [Ref. biometric-2]. • `ansiFmd` - ANSI FMD template format [Ref. biometric-3]. • `qso` - Raw QSO image format. • `wso` - WSQ image format. • `reservedRaw1` - Reserved for a vendor-defined Raw format. • `reservedTemplate1` - Reserved for a vendor-defined Template format. • `reservedRaw2` - Reserved for a vendor-defined Raw format. • `reservedTemplate2` - Reserved for a vendor-defined Template format. • `reservedRaw3` - Reserved for a vendor-defined Raw format. • `reservedTemplate3` - Reserved for a vendor-defined Template format.
type/algorithm Specifies the encryption algorithm. The following values are possible: • `ecb` - Triple DES with Electronic Code Book. • `cbc` - Triple DES with Cipher Block Chaining • `cfb` - Triple DES with Cipher Feed Back. • `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 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. The following values are possible: • `scan` - Scan data only, for example to enroll a user or collect data for matching in an external biometric system. • `match` - 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	Yes
`"format": "isoFid",`	string	Yes

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Payload	Type	Required
`"algorithm": "ecb",`	string	Yes
`"keyName": Add example to YAML`	string	
`},`		
`"data": Add example to YAML`	string	Yes
`}]`		
`}`		

Properties

completionCode
The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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 dataFormats capability field.
- `keyNotFound` - The specified key name is not found.

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. The following values are possible:

- `isoFid` - Raw ISO FID format [Ref. biometric-4].
- `isoFmd` - ISO FMD template format [Ref. biometric-5].
- `ansiFid` - Raw ANSI FID format [Ref. biometric-2].
- `ansiFmd` - ANSI FMD template format [Ref. biometric-3].
- `qso` - Raw QSO image format.
- `wso` - WSQ image format.
- `reservedRaw1` - Reserved for a vendor-defined Raw format.
- `reservedTemplate1` - Reserved for a vendor-defined Template format.
- `reservedRaw2` - Reserved for a vendor-defined Raw format.
- `reservedTemplate2` - Reserved for a vendor-defined Template format.
- `reservedRaw3` - Reserved for a vendor-defined Raw format.
- `reservedTemplate3` - Reserved for a vendor-defined Template format.

dataList/type/algorithm
Specifies the encryption algorithm. The following values are possible:

- `ecb` - Triple DES with Electronic Code Book.
- `cbc` - Triple DES with Cipher Block Chaining
- `cfb` - Triple DES with Cipher Feed Back.
- `rsa` - RSA Encryption.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
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 individual binary data stream encoded in base64.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

Event Messages

- Biometric.PresentSubjectEvent
- Biometric.SubjectDetectedEvent
- Biometric.RemoveSubjectEvent
- Biometric.SubjectRemovedEvent
- Biometric.DataClearedEvent
- Biometric.OrientationEvent

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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
`{`		
`"timeout": 5000,`	integer	
`"dataList": [{`	array (object)	
`"type": {`	object	Yes
`"format": "isoFid",`	string	Yes
`"algorithm": "ecb",`	string	Yes
`"keyName": Add example to YAML`	string	
`},`		
`"data": Add example to YAML`	string	Yes
`}]`		
`}`		

Properties

timeout
Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be canceled.
default: 0

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. The following values are possible:

- `isoFid` - Raw ISO FID format [Ref. biometric-4].
- `isoFmd` - ISO FMD template format [Ref. biometric-5].
- `ansiFid` - Raw ANSI FID format [Ref. biometric-2].
- `ansiFmd` - ANSI FMD template format [Ref. biometric-3].
- `qso` - Raw QSO image format.
- `wso` - WSQ image format.
- `reservedRaw1` - Reserved for a vendor-defined Raw format.
- `reservedTemplate1` - Reserved for a vendor-defined Template format.
- `reservedRaw2` - Reserved for a vendor-defined Raw format.
- `reservedTemplate2` - Reserved for a vendor-defined Template format.
- `reservedRaw3` - Reserved for a vendor-defined Raw format.
- `reservedTemplate3` - Reserved for a vendor-defined Template format.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
dataList/type/algorithm Specifies the encryption algorithm. The following values are possible: • `ecb` - Triple DES with Electronic Code Book. • `cbc` - Triple DES with Cipher Block Chaining • `cfb` - Triple DES with Cipher Feed Back. • `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 individual binary data stream encoded in base64.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "invalidData",`	string	
`"storageList": [{`	array (object)	
`"identifier": 0,`	integer	Yes
`"type": {`	object	Yes
`"format": "isoFid",`	string	Yes
`"algorithm": "ecb",`	string	Yes
`"keyName": Add example to YAML`	string	
`}`		
`}]`		
`}`		

Properties
completionCode The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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 dataFormats 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 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/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. The following values are possible: • `isoFid` - Raw ISO FID format [Ref. biometric-4]. • `isoFmd` - ISO FMD template format [Ref. biometric-5]. • `ansiFid` - Raw ANSI FID format [Ref. biometric-2]. • `ansiFmd` - ANSI FMD template format [Ref. biometric-3]. • `qso` - Raw QSO image format. • `wso` - WSQ image format. • `reservedRaw1` - Reserved for a vendor-defined Raw format. • `reservedTemplate1` - Reserved for a vendor-defined Template format. • `reservedRaw2` - Reserved for a vendor-defined Raw format. • `reservedTemplate2` - Reserved for a vendor-defined Template format. • `reservedRaw3` - Reserved for a vendor-defined Raw format. • `reservedTemplate3` - Reserved for a vendor-defined Template format.
storageList/type/algorithm Specifies the encryption algorithm. The following values are possible: • `ecb` - Triple DES with Electronic Code Book. • `cbc` - Triple DES with Cipher Block Chaining • `cfb` - Triple DES with Cipher Feed Back. • `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

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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 supports *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 supports *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* is 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* is 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
`{`		
`"timeout": 5000,`	integer	
`"compareMode": "verify",`	string	
`"identifier": 0,`	integer	
`"maximum": 0,`	integer	
`"threshold": 0`	integer	
`}`		

© 2021 CEN 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 canceled. default: 0
compareMode Specifies the type of match operation that is being done. The following values are possible: • `verify` - The biometric data will be compared as a one to one verification operation. • `identity` - The biometric data will 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 property can be omitted.
maximum Specifies the maximum number of matches to return. In the case where *compareMode* is verify this property can be omitted.
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	Yes
`"identifier": 0,`	integer	Yes
`"data": {`	object	Yes
`"type": {`	object	Yes
`"format": "isoFid",`	string	Yes
`"algorithm": "ecb",`	string	Yes
`"keyName": Add example to YAML`	string	
`},`		
`"data": Add example to YAML`	string	Yes
`}`		
`}]`		
`}`		

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
completionCode The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
errorCode Specifies the error code if applicable. The following values are possible: • `noImportedData` - The command failed because no data was imported previously using the Biometric.Import 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. • `invalidThreashold` - 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. This property can be omitted 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. The following values are possible: • `isoFid` - Raw ISO FID format [Ref. biometric-4]. • `isoFmd` - ISO FMD template format [Ref. biometric-5]. • `ansiFid` - Raw ANSI FID format [Ref. biometric-2]. • `ansiFmd` - ANSI FMD template format [Ref. biometric-3]. • `qso` - Raw QSO image format. • `wso` - WSQ image format. • `reservedRaw1` - Reserved for a vendor-defined Raw format. • `reservedTemplate1` - Reserved for a vendor-defined Template format. • `reservedRaw2` - Reserved for a vendor-defined Raw format. • `reservedTemplate2` - Reserved for a vendor-defined Template format. • `reservedRaw3` - Reserved for a vendor-defined Raw format. • `reservedTemplate3` - Reserved for a vendor-defined Template format.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
templateList/data/type/algorithm Specifies the encryption algorithm. The following values are possible: • `ecb` - Triple DES with Electronic Code Book. • `cbc` - Triple DES with Cipher Block Chaining • `cfb` - Triple DES with Cipher Feed Back. • `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 individual binary data stream encoded in base64.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

Event Messages

- Biometric.DataClearedEvent

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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 unsupportedCommand 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
`{`		
`"timeout": 5000,`	integer	
`"compareMode": "verify",`	string	
`"identifier": 0,`	integer	
`"maximum": 0,`	integer	
`"threshold": 0`	integer	
`}`		

Properties

timeout
Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be canceled.
default: 0

compareMode
Specifies the type of match operation that is being done. The following values are possible:

- `verify` - The biometric data will be compared as a one to one verification operation.
- `identity` - The biometric data will 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 property can be omitted.

maximum
Specifies the maximum number of matches to return. In the case where *compareMode* is verify this property can be omitted.

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	

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Payload	Type	Required
`"errorCode": "invalidIdentifier"`	string	
`}`		

Properties

completionCode

The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription

If included, this contains additional vendor dependent information to assist with problem resolution.

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

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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
`{`		
`"timeout": 5000,`	integer	
`"clearData": "scannedData"`	string	
`}`		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be canceled.

default: 0

clearData

This parameter indicates the type of data to be cleared from storage. If this property is omitted, then all stored data will be cleared.

- `scannedData` - Raw image data that has been scanned using the Biometric.Read command can be cleared.
- `importedData` - Template data that was imported using the Biometric.Import command can be cleared.
- `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

The completion code.

errorDescription

If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

- Biometric.DataClearedEvent

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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": "scannedData"`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

clearData
This parameter indicates the type of data to be cleared from storage. If this property is omitted, then all stored data will be cleared.

- `scannedData` - Raw image data that has been scanned using the Biometric.Read command can be cleared.
- `importedData` - Template data that was imported using the Biometric.Import command can be cleared.
- `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
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

- Biometric.DataClearedEvent

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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
`{`		
`"timeout": 5000,`	integer	
`"persistanceMode": "persist"`	string	
`}`		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be canceled.

default: 0

persistanceMode

Specifies the data persistence mode. This controls how biometric data that has been captured using the Biometric.Read command will persist. This value itself is persistent. The following values are possible:

- `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.
- `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

The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription

If included, this contains additional vendor dependent information to assist with problem resolution.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
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

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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": "scannedData"`	string	Yes
`}`		

Properties

clearData

This parameter indicates the type of data to be cleared from storage. If this property is omitted, then all stored data will be cleared.

- `scannedData` - Raw image data that has been scanned using the Biometric.Read command can be cleared.
- `importedData` - Template data that was imported using the Biometric.Import command can be cleared.
- `setMatchedData` - Match criteria data that was set using the Biometric.Match command can be cleared.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

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.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

17. Camera Interface

This chapter defines the Camera interface functionality and messages.

17.1 Summary

This specification describes the functionality of the services provided by the Camera services under XFS. Banking camera systems usually consist of a recorder, a video mixer and one or more cameras. If there are several cameras, each camera focuses a special place within the self-service area (e.g. the room, the customer or the cash tray). By using the video mixer it can be decided, which of the cameras should take the next photo. Furthermore data can be given to be inserted in the photo (e.g. date, time or bank code). If there is only one camera that can switch to take photos from different positions, it is presented by the Service Provider as a set of cameras, one for each of its possible positions.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

17.2 Command Messages

17.2.1 Camera.TakePicture

This command is used to start the recording of the camera system. It is possible to select which camera or which camera position should be used to take a picture. Data to be displayed on the photo can be specified using the camData parameter.

Command Message

Payload	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"camera": "room",`	string	
`"camData": {`	object	
`"exampleProperty1": Add example to YAML,`	string	
`"exampleProperty2": Add example to YAML`	string	
`},`		
`"pictureFile": 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 canceled.

default: 0

camera

Specifies the camera that should take the photo as one of the following values:

- `room` - Monitors the whole self-service area.
- `person` - Monitors the person standing in front of the self-service machine.
- `exitSlot` - Monitors the exit slot(s) of the self-service machine.

camData

Specifies the text string to be displayed on the photo. If the maximum text length is exceeded it will be truncated. In this case or if the text given is invalid, a Camera.InvalidDataEvent event will be generated. Nevertheless the picture is taken.

camData/exampleProperty1 (example name)

pictureFile

Specifies the full path and file name of the images to be taken by a camera device. The file name includes the image format specific the extension. The Service Provider is responsible for converting the image into the required format.

This property can be omitted, if the Common.Capabilities.pictureFile property is false.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	string	
`}`		

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

- Camera.MediaThresholdEvent
- Camera.InvalidDataEvent

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

17.2.2 Camera.Reset

This command is used by the client to perform a hardware reset which will attempt to return the camera device to a known good 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 canceled.
default: 0

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

17.3 Event Messages

17.3.1 Camera.InvalidDataEvent

This event is used to specify that the text string given was too long or in some other way invalid.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

17.4 Unsolicited Messages

17.4.1 Camera.MediaThresholdEvent

This event is used to specify that the state of the recording media reached a threshold.

Payload	Type	Required
`{`		
`"mediaThreshold": "ok"`	string	
`}`		

Properties

mediaThreshold

Specified as one of the following.

- `ok` - The recording media is a good state.
- `high` - The recording media is almost full.
- `full` - The recording media is full.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

18. German DK(ZKA) Interface

This chapter defines the German 'Geldkarte' functionality and messages.

18.1 Summary

This country specific interface is able to handle the German 'Geldkarte', which is an electronic purse specified by the DK (Deutsche Kreditwirtschaft) formerly known as the ZKA (Zentraler Kreditausschuß) protocol.

18.2 General Information

18.2.1 References

ID	Description
dk-6	ZKA / Bank-Verlag, Köln, Schnittstellenspezifikation für die ec-Karte mit Chip, Online-Personalisierung von Terminal-HSMs, Version 3.0, 2. 4. 1998
dk-7	ZKA / Bank-Verlag, Köln, Schnittstellenspezifikation für die ZKA-Chipkarte, Online-Vor-Initialisierung und Online-Anzeige einer Außerbetriebnahme von Terminal-HSMs, Version 1.0, 04.08.2000
dk-8	473x Programmers Reference Volume 1 - TP-820399-001A.
dk-13	473x Log Reference Guide - TP-820398-001A.
dk-14	Diebold‘s Specification for support of Online Preinitialization and Personalization of Terminal HSMs (OPT) and support for the PAC/MAC standards for the 473x Protocol, Diebold USA, Revision 1.10, revised on May 2002
dk-17	German ZKA specification, published by: Bank-Verlag Koeln
dk-27	Regelwerk für das deutsche ec-Geldautomaten-System, Stand: 22. Nov. 1999
dk-30	dvg Hannover, Schnittstellenbeschreibung für Autorisierungsanfragen bei internationalen Verfügungen unter Verwendung der Spur 2, Version 2.6, Stand: 30.03.2000
dk-31	ZKA / Bank-Verlag, Köln,.Schnittstellenspezifikation für die ec-Karte mit Chip, Geldkarte Ladeterminals, Version 3.0, 2. 4. 1998
dk-34	ZKA / Bank-Verlag, Köln, Schnittstellenspezifikation für die ec-Karte mit Chip PIN-Änderungsfunktion, Version 3.0, 12.05.1999
dk-37	PCI Security Standards Council PCI PTS approval list

18.2.2 German ZKA GeldKarte (Deutsche Kreditwirtschaft)

The pin service is able to handle the German "Geldkarte", which is an electronic purse specified by the DK (Deutsche Kreditwirtschaft) formerly known as the ZKA (Zentraler Kreditausschuß) protocol. For anyone attempting to write an application that handles this type of chip card, it is essential to read and understand the ZKA specifications. See [Ref. dk-17], [Ref. dk-6] and [Ref. dk-7].

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

How to use the SecureMsg commands

This is to describe how an application should use the DK.SecureMsgSend and DK.SecureMsgReceive commands for transactions involving chipcards with a German ZKA GeldKarte chip.

- Applications must call DK.SecureMsgSend for every command they send to the chip or to a host system, including those commands that do not actually require secure messaging. This enables the Service Provider to remember security-relevant data that may be needed or checked later in the transaction.

- Applications must pass a complete message as input to DK.SecureMsgSend, with all properties - including those that will be filled by the Service Provider - being present in the correct length. All properties that are not filled by the Service Provider must be filled with the ultimate values in order to enable MACing by the Service Provider.

- Every command DK.SecureMsgSend that an application issues must be followed by exactly one command DK.SecureMsgReceive that informs the Service Provider about the response from the chip or host. If no response is received (timeout or communication failure) the application must issue a DK.SecureMsgReceive command with msg property omitted to inform the Service Provider about this fact.

- If a system is restarted after a DK.SecureMsgSend was issued to the Service Provider but before the DK.SecureMsgReceive was issued, the restart has the same effect as a DK.SecureMsgReceive command with msg property omitted.

- Between a DK.SecureMsgSend and the corresponding DK.SecureMsgReceive no DK.SecureMsgSend with the same Protocol must be issued. Other commands from the KeyManagement, PinPad, Crypto, Keyboard including DK interfaces accessing to the same device.

Protocol isoAs

This protocol handles ISO8583 messages between an ATM and an authorization system (AS).

Only messages in the new ISO format, with new PAC/MAC-format using session keys and Triple-DES are supported.

Authorization messages may be used to dispense the amount authorized in cash or to load the amount into an electronic purse (GeldKarte).

For loading a GeldKarte the only type of authorization supported is a transaction originating from track 3 of a German ec-card (message types 0200/0210 for authorization and 0400/0410 for reversal).

For dispensing cash, transactions originating from international cards (message types 0100/0110 and 0400/0410) are supported as well.

The following bitmap positions are filled by the Service Provider:

- BMP11 - Trace-Nummer
- BMP52 - PAC
- BMP57 - Verschlüsselungsparameter (only the challenge values RNDMES and RNDPAC)
- BMP64 - MAC

These bitmaps have to be present and the corresponding flag has to be set in the primary bitmap when the ISO message is passed to the HSM.

The following bitmap positions are checked by the Service Provider and have to be filled by the application:

- Nachrichtentyp
- BMP3 - Abwicklungskennzeichen (only for GeldKarte, not for cash)
- BMP4 - Transaktionsbetrag (only for GeldKarte, not for cash)
- BMP41 - Terminal-ID
- BMP42 - Betreiber-BLZ

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

For additional documentation of authorization messages see [Ref. dk-27] – [Ref. dk-30].

Protocol isoLz

This protocol handles ISO8583 messages between a "Ladeterminal" and a "Ladezentrale" (LZ).

Only messages in the new ISO format, with new MAC-format using session keys and Triple-DES are supported.

Both types of GeldKarte chip (type 0 = DEM, type 1 = EUR) are supported.

The following bitmap positions are filled by the Service Provider:

- BMP11: Trace-Nummer
- BMP57: Verschlüsselungsparameter (only the challenge value RNDMES)
- BMP64: MAC

These bitmaps have to be present and the corresponding flag has to be set in the primary bitmap when the ISO message is passed to the HSM.

The following bitmap positions are checked by the Service Provider and have to be filled by the application:

- Nachrichtentyp
- BMP3: Abwicklungskennzeichen
- BMP4: Transaktionsbetrag
- BMP12: Uhrzeit
- BMP13: Datum
- BMP25: Konditionscode
- BMP41: Terminal-ID
- BMP42: Betreiber-BLZ (caution: "Ladeentgelt" also in BMP42 is not set by the EPP)
- BMP61: Online-Zeitpunkt
- BMP62: Chipdaten

The following bitmap positions are only checked if they are available:

- BMP43: Standort
- BMP60: Kontodaten Ladeterminal

For a documentation of the Ladezentrale interface see [Ref. dk-31].

Protocol isoPs

This protocol handles ISO8583 messages between a terminal and a "Personalisierungsstelle" (PS). These messages are about OPT.

The Service Provider creates the whole message with DK.SecureMsgSend, including message type and bitmap.

For a documentation of the Personalisierungsstelle interface see [Ref. dk-7].

Protocol chipZka

This protocol is intended to handle messages between the application and a GeldKarte.

Both types of GeldKarte are supported.

Both types of load transactions ("Laden vom Kartenkonto" and "Laden gegen andere Zahlungsmittel") are supported.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

See the chapter "Command Sequence" below for the actions that Service Providers take for the various chip card commands.

Only the command APDUs to and the response APDUs from the chip must be passed to the Service Provider, the ATR (answer to reset) data from the chip is not passed to the Service Provider.

For a documentation of the chip commands used to load a GeldKarte see [Ref. dk-31].

Protocol pbm

This protocol handles host messages between a terminal and a host system, as specified by PBM protocol.

For documentation of this protocol see [Ref. dk-8] – [Ref. dk-13].

Some additions are defined to the PBM protocol in order to satisfy the German ZKA 3.0 PAC/MAC standard. See [Ref. dk-14].

The commands DK.SecureMsgSend and DK.SecureMsgReceive handle the PAC and MAC in the VARDATA 'K' or 'Q' subfield of transactions records and responses. The MAC in the traditional MACODE field is not affected.

In order to enable the Service Provider to understand the messages, the application must provide the messages according to the following rules:

- All alphanumeric fields must be coded in EBCDIC.
- Pre-Edit (padding and blank compression) must not be done by the application. The Service Provider will check the macMode field and will perform the pre-edit according to what the macMode field intends.
- In order to enable the Service Provider to find the vardata subfield 'K' or 'Q', it must be included in the message by the application, with the indicator 'K' or 'Q' and its length set.
- Because CARDDATA (track 2) and T3DATA (track 3) fields always take part in the MAC computation for a transaction record, these fields must be included in the message, even if they already have been sent to the host in a previous transaction record and the CI-Option shortRec prevents them from being sent again.

Protocol hsmLdi

With this protocol an application can request information about the personalized OPT groups.

The information returned consists of personalization record like in BMP62 of an OPT response but without MAC.

Data format:

XX XX VV - group ID and version number (BCD format)

XX - number of LDIs within the group (BCD format) … first LDI of the group … last LDI of the group

XX XX VV - group ID and version number (BCD format) ... etc. for several groups

Each LDI consists of:

NN Number of the LDI

00 Alg. Code

LL Length of the following data

XX...XX data of the LDI

For each group ID the Service Provider must always return the standard LDI. LDI 01 must also be returned for groups AF XX VV. Further LDIs can be returned optionally.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Protocol genas

This protocol provides the capability to create a PAC (encrypted PIN block) and to create and verify a MAC for a proprietary message. As the Service Provider does not know the message format, it cannot complete the message by adding security relevant fields like random values, PAC and MAC, like it does for the protocol isoAs. Only the application is able to place these fields into the proper locations. Using this protocol, an application can generate the PAC and the random values in separate steps, adds them to the proprietary send-message, and finally lets the Service Provider generate the MAC. The generated MAC can then be added to the send-message as well.

For a received message, the application extracts the MAC and the associated random value and passes them along with the entire message data to the Service Provider for MAC verification.

PAC generation supports PIN block ISO-Format 0 and 1 for 3DES and ISO-Format 4 for AES.

Command description:

The first byte of field msg of secMsg contains a subcommand, which is used to qualify the type of operation. The remaining bytes of the command data are depending on the value of the subcommand.

The following sub-commands are defined:

- generatePAC 3DES (Code 0x01) Returns the encrypted PIN block together with generation and version values of the Master Key and the PAC random value.
- getMacRandom 3DES (Code 0x02) Returns the generation and version values of the Master Key and the MAC random value.
- GenerateMAC 3DES (Code 0x03) Returns the generated MAC for the message data passed in. Note that the MAC is generated for exactly the data that is presented (contents and sequence). Data that should not go into MAC calculation must not be passed in.
- verifyMac 3DES (Code 0x04) Generates a MAC for the data passed in and compares it with the provided MAC value. MAC random value, key generation and key version must be passed in separately.
- Generate PAC AES (Code 0x05) Returns the encrypted PIN block wrapped in the BMP110.2 (Dataset 01).
- Get MAC Random AES (Code 0x06) Returns the MAC random value wrapped in the BMP110.3 (Dataset 02).
- Generate MAC AES (Code 0x07) Returns the generated MAC for the message data passed in. Note that the MAC is generated for exactly the data that is presented (contents and sequence). Data that should not go into MAC calculation must not be passed in. Used algorithm is CMAC.
- Verify MAC AES (Code 0x08) Generates a MAC for the data passed in and compares it with the provided MAC value. The MAC data must be passed in as BMP110.3 (Dataset 02) in the format: 08 (sub-command) + BMP110.3 + MAC + message to be verified.

Command/Message sequence:

Command	msg in SecureMsgCommand	msg in SecureMsgCompletion	Service Provider's actions
DK.SecureMsgSend	Byte 0: 0x01 (Generate PAC) Byte 1: format (0 or 1) Byte 2-9: ANF (Primary Account Number, if length is less than 12 digits, value must be left padded with binary 0, only applicable for format 0)	Byte 0: key generation Byte 1: key version Byte 2-17: PAC random Byte 18-25: PAC value (all values are binary values)	Generates a session key for PAC generation and finally the PAC itself. Determine generation and version values of Master- Key and return them along with the random value

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Command	msg in SecureMsgCommand	msg in SecureMsgCompletion	Service Provider's actions
DK.SecureMsgSend	Byte 0: 0x02 (Get MAC Random)	Byte 0: key generation Byte 1: key version Byte 2-17: MAC random (all values are binary values	Generates a session key for MAC generation (see next step below) Determine generation and version values of Master- Key and return them along with the random value
DK.SecureMsgSend	Byte 0: 0x03 (Generate MAC) Byte 1-n: Message to be mac'ed (all values are binary values)	Byte 0-7: generated MAC (binary value)	Generates MAC over bytes 1-n of the inbound message using the session key created in the previous step.
DK.SecureMsgReceive	Byte 0: 0x04 (Verify MAC) Byte 1: key generation Byte 2: key version Byte 3-18: MAC random Byte 19-26: MAC Byte 27-n: Message to be verified (all values are binary values) NOTE: If no message has been received, this function must be called by omitting Bytes 1-n	N/A	Generates a session key using the Master key identified by key generation and version by using the random value passed in. Generates a MAC for the message data passed in and compare the resulting MAC with the MAC passed in.
DK.SecureMsgSend	Byte 0: 0x05 (Generate PAC AES) Byte 1: format (4)	Byte 0: 01 Identification for Dataset 01 Byte 1-2: length of data Byte 3-n: data	Generates a session key for PAC generation and finally the PAC itself. Returned values are in the format of dataset 01 of BMP110
DK.SecureMsgSend	Byte 0: 0x06 (Get MAC Random AES)	Byte 0: 02 Identification for Dataset 02 Byte 1-2: length of data Byte 3-n: data	Generates a session key for MAC generation (see next step below) Returned values are in the format of dataset 02 of BMP110

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Command	msg in SecureMsgCommand	msg in SecureMsgCompletion	Service Provider's actions
DK.SecureMsgSend	Byte 0: 0x07 (Generate MAC AES) Byte 1-n: Message to be mac'ed (all values are binary values)	Byte 0-7: generated MAC (binary value)	Generates MAC over bytes 1-n of the inbound message using the session key created in the previous step.
DK.SecureMsgReceive	Byte 0: 0x08 (Verify MAC AES) Byte 1-37: BMP110 Dataset 02 Byte 38-45: MAC Byte 46-n: Message to be verified (all values are binary values)	N/A	Generates a session key using the Master key identified by key generation and version by using the random value passed in. Generates a MAC for the message data passed in and compare the resulting MAC with the MAC passed in.

Returns:
The error code formatInvalid is returned when:

- The subcommand in Byte 0 of msg for Execute Command DK.SecureMsgSend with protocol genas is not 01, 02, 03, 05, 06 or 07.
- The subcommand in Byte 0 of msg for Execute Command DK.SecureMsgReceive with protocol genas is not 04 or 08.
- The subcommand in Byte 0 of msg for Execute Command DK.SecureMsgSend with protocol genas is 01 and Byte 1 is not 00 and not 01 (PIN block format is not ISO-0 and ISO-1).
- The subcommand in Byte 0 of msg for Execute Command DK.SecureMsgSend with protocol genas is 05 and Byte 1 is not 04 (PIN block format is not ISO-4)
- The individual command data length for a subcommand is less than specified.

The error code hsmStateInvalid is returned when:

- The subcommand in Byte 0 of msg for Execute Command DK.SecureMsgSend with protocol genas is 03 (Generate MAC) without a preceding getMacRandom (secureMsgSend with subcommand 02).
- The subcommand in Byte 0 of msg for Execute Command DK.SecureMsgSend with protocol genas is 07 (Generate MAC) without a preceding getMacRandom (secureMsgSend with subcommand 06).

The error code macInvalid is returned when:

- The subcommand in Byte 0 of msg for Execute Command DK.SecureMsgReceive with protocol genas is 04 (Verify MAC) and the MACs did not match.

The error code keyNotFound is returned when:

- The subcommand in Byte 0 of msg for Execute Command DK.SecureMsgSend with protocol genas is 01 or 05 (Generate PAC) and the Service Provider does not find a master key.
- The subcommand in Byte 0 of msg for Execute Command DK.SecureMsgSend with protocol genas is 02 or 06 (Get MAC Random) and the Service Provider does not find a master key.
- The subcommand in Byte 0 of msg for Execute Command DK.SecureMsgReceive with protocol genas is 04 or 08 (Verify MAC) and the Service Provider does not find a key for the provided key generation and key version values.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

The error code noPin is returned when:

- The subcommand in Byte 0 of msg for Execute Command DK.SecureMsgSend with protocol genas is 01 or 05 (Generate PAC) and no PIN or insufficient PIN-digits have been entered.

Protocol chipPinChg

This protocol is intended to handle messages exchanged between the PIN pad and a GeldKarte, which are all related to the PIN change transaction.

Only Type-1-GeldKarte is supported, because the former Type-0-GeldKarte will no longer be used as it was a dedicated Deutsche Mark electronic purse only. The Type-1-GeldKarte is used for Euro currency.

The transaction types supported are:

- PIN-Activation ("PIN-Aktivierung")
- PIN-Activation after Failure ("PIN-Aktivierung nach Fehlerfall")
- PIN-Change ("PIN-Änderung")

See the command sequence section below for the actions that Service Providers take for the various chip card commands.

Only the command APDUs to and the response APDUs from the chip must be passed to the Service Provider, the ATR (answer to reset) data from the chip is not passed to the Service Provider.

For the complete documentation of the chip commands used for PIN-Change see [Ref. dk-34].

Protocol pinCmp

This simple protocol is used to perform a comparison of two PINs entered into the PIN Pad. In order to be able to compare the PINs, the first value must be temporary stored while the second value is entered. The user will be prompted to enter the PIN twice. After the PIN has been entered for the first time, the PIN pad needs to store the PIN value into a temporary location. After the user has entered the PIN for the second time, the PIN pad has to compare both values.

This protocol consists of two subcommands. The first subcommand requests the PIN pad to save the PIN value entered by the Keyboard.PinEntry command for subsequent comparison. The second subcommand forces the PIN pad to compare the PIN stored with the second value entered by the Keyboard.PinEntry command. The status of the PIN comparison is returned in the output data.

See the command sequence section below for the actions that Service Providers take for this protocol.

Use of pinCmp with non-GeldKarte ZKA PIN Management

For use with the non-GeldKarte ZKA PIN compare function (see [Ref. dk-37)] there are two more subcommands "start PIN compare" and "end PIN compare". These have to be called before entry of the first PIN and after querying of the PAC to signal the end of the PIN comparison, respectively.

This is the command sequence for the non-GeldKarte transaction:

Flow	Command pin	protocol	msg in SecureMsgCommand	msg in SecureMsgCompletion	Service Provider's actions
PIN Compare					

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Flow	Command pin	protocol	msg in SecureMsgCommand	msg in SecureMsgCompletion	Service Provider's actions
Start PIN comparision	DK.SecureMsgSend	pinCmp	Byte 0: 0x00 (Start PIN compare)		Prepare EPP for PIN comparison. Output data buffer length is zero.
Let the user enter the new PIN for the first time.	Keyboard.PinEntry	N/A	N/A	N/A	PIN entry.
	DK.SecureMsgSend	pinCmp	Byte 0: 0x01 (Save PIN)		Save the PIN value entered for subsequent compare. Output data buffer length is zero.
Let the user enter the new PIN for the second time	Keyboard.PinEntry	N/A	N/A	N/A	PIN entry.
	DK.SecureMsgSend	pinCmp	Byte 0: 0x02 (Compare PINs)	Byte 0: 0x00 when PIN does not match, and 0x01 when PIN does match.	Compare PIN values.
Get the PAC of the new PIN via genas or isoAs (as usual).					
End PIN comparison.	DK.SecureMsgSend	pinCmp	Byte 0: 0xFF (End PIN compare)		All PIN buffers are cleared. Output data buffer length is zero.

Please note that no other PIN commands apart from Keyboard.PinEntry and DK.SecureMsgSend as specified above are allowed inside a start / end PIN compare flow, with the exception of creating the PAC for the old PIN. While the old PIN always has to be entered (using Keyboard.PinEntry from Keyboard interface) before the "Start PIN Compare", the PAC for the old PIN may be created (using DK.SecureMsgSend with protocol=genas) after the "Start

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

PIN Compare" if (enforced by the host protocol) the same session key SKPAC has to be used for encrypting both the old and the new PIN.

Protocol isoPinChg

This protocol handles ISO8583 messages between an ATM and an authorization system (AS) related to the transactions:

- PIN-Activation ("PIN-Aktivierung")
- PIN-Activation after Failure ("PIN-Aktivierung nach Fehlerfall")
- PIN-Change ("PIN-Änderung")

The message types supported are:

- 0640 (PIN Change / PIN Activation Request)
- 0642 (Confirmation / Reversal Request for PIN Change / PIN Activation)
- 0643 (Confirmation Repeat Request for PIN Change / PIN Activation)
- 0650 (PIN Change / PIN Activation Response)
- 0652 (Confirmation / Reversal Response)

The following bitmap positions are filled by the Service Provider:

- BMP52 PAC
- BMP57 Verschlüsselungsparameter (KTerminal Generation, KTerminal Version, RNDMES and RNDPAC)
- BMP62 (EFID, EFINFO, Record number of PIN, Key Version of KCard, EFFBZ, PAC, Random value returned by getChallenge)
- BMP64 MAC

These bitmaps have to be present and the corresponding flag has to be set in the primary bitmap when the ISO message is passed to the HSM.

See the command sequence section below for the actions that Service Providers take for the various messages.

For the complete documentation of the messages used for PIN-Change see [Ref. dk-34].

Command Sequence

The following list shows the sequence of actions an application has to take for the various GeldKarte Transactions. Please note that this is a summary and is just intended to clarify the purpose of the chipcard-related ... commands. In no way it can replace the ZKA specifications mentioned above.

Command	protocol	msg	Service Provider's actions
Preparation for Load/Unload			
DK.SecureMsgSend	chipZka	Command APDU SELECT FILE DFBÖRSE	
DK.SecureMsgReceive	chipZka	Response APDU	recognize type of chip
DK.SecureMsgSend	chipZka	Command APDU READ RECORD EFID	
DK.SecureMsgReceive	chipZka	record EFID	store EFID
DK.SecureMsgSend	chipZka	Command APDU READ RECORD EFLLOG	
DK.SecureMsgReceive	chipZka	record EFLLOG	

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Command	protocol	msg	Service Provider's actions
DK.SecureMsgSend	chipZka	Command APDU READRECORD EFBÖRSE	
DK.SecureMsgReceive	chipZka	record EFBÖRSE	
DK.SecureMsgSend	chipZka	Command APDU READRECORD EFBETRAG	
DK.SecureMsgReceive	chipZka	record EFBETRAG	
Load against other ec-Card			
DK.SecureMsgSend	chipZka	**for type 0 chips only** Command APDU READ RECORD EFKEYD	
DK.SecureMsgReceive	chipZka	record EFKEYD	
DK.SecureMsgSend	chipZka	**for type 1 chips only** Command APDU GET KEYINFO	
DK.SecureMsgReceive	chipZka	Response APDU	
DK.SecureMsgSend	chipZka	Command APDU GET CHALLENGE	
DK.SecureMsgReceive	chipZka	Random number RND1 from Chip	store RND1
DK.SecureMsgSend	chipZka	Command APDU LADEN EINLEITEN with Secure msg.	fill: -Terminal ID -Traceno. -RND2 -MAC
DK.SecureMsgReceive	chipZka	Response APDU	store response APDU for later check of isoLz message, BMP 62
DK.SecureMsgSend	isoAz	ISO8583 message 0200 Authorization Request	Fill: - Traceno. (BMP 11) - PAC (BMP 52) - RNDMES + RNDPAC (BMP 57) - MAC (BMP 64) check other security relevant fields
DK.SecureMsgReceive	isoAz	ISO8583 message 0210 Authorization Response	check MAC and other security relevant fields
DK.SecureMsgSend	isoLz	ISO8583 message 0200 Ladeanfrage	Fill: - Traceno. (BMP 11) - RNDMES (BMP 57) - MAC (BMP 64) check other security relevant fields.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Command	protocol	msg	Service Provider's actions
DK.SecureMsgReceive	isoLz	ISO8583 message 0210 Ladeantwort	check MAC and other security relevant fields, store BMP62 for later use in LADEN command.
DK.SecureMsgSend	chipZka	Command APDU GET CHALLENGE	
DK.SecureMsgReceive	chipZka	Random number RND3 from chip	store RND3
DK.SecureMsgSend	chipZka	Command APDU LADEN with Secure msg.	provide complete command from BMP62 of isoLz response, compute command MAC
DK.SecureMsgReceive	chipZka	Response APDU	check response MAC
DK.GetJournal	isoLz	Vendor specific	
DK.GetJournal	IsoAz	Vendor specific	
Reversal of a Load against other ec-Card			
DK.SecureMsgSend	chipZka	Command APDU SELECT FILE DFBÖRSE	
DK.SecureMsgReceive	chipZka	Response APDU	
DK.SecureMsgSend	chipZka	Command APDU GET CHALLENGE	
DK.SecureMsgReceive	chipZka	Random number RND5 from chip	store RND5
DK.SecureMsgSend	chipZka	Command APDU LADEN EINLEITEN with Secure msg.	Fill: -Terminal ID -Traceno. -RND6 -Keyno. KGKLT -MAC
DK.SecureMsgReceive	chipZka	Response APDU	store response APDU for later check of isoLz message, BMP 62
DK.SecureMsgSend	IsoAz	ISO8583 message 0400 Storno	Fill: - Traceno. (BMP 11) - PAC (BMP 52) - RNDMES + RNDPAC (BMP 57) - MAC (BMP 64) check other security relevant fields

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Command	protocol	msg	Service Provider's actions
DK.SecureMsgReceive	IsoAz	ISO8583 message 0410 Storno Response	check MAC and other security relevant fields.
DK.SecureMsgSend	isoLz	ISO8583 message 0400 Storno	Fill: - Traceno. (BMP 11) - RNDMES (BMP 57) - MAC (BMP 64) check other security relevant fields.
DK.SecureMsgReceive	isoLz	ISO8583 message 0410 Storno Response	check MAC and other security relevant fields, store BMP62 for later use in LADEN command.
DK.SecureMsgSend	chipZka	Command APDU GET CHALLENGE	
DK.SecureMsgReceive	chipZka	Random number RND7 from chip	store RND7
DK.SecureMsgSend	chipZka	Command APDU LADEN with Secure msg.	provide complete command from BMP62 of isoLz response, compute command MAC
DK.SecureMsgReceive	chipZka	Response APDU	check response MAC
DK.GetJournal	isoLz	Vendor specific	
DK.GetJournal	IsoAz	Vendor specific	
PIN Verification Type 0			
DK.SecureMsgSend	chipZka	Command APDU GET CHALLENGE	
DK.SecureMsgReceive	chipZka	Random number RND0 from chip	store RND0
DK.SecureMsgSend	chipZka	Command APDU EXTERNAL AUTHENTICATE	fill -Keyno. KINFO - ENCRND
DK.SecureMsgReceive	chipZka	Response APDU	
DK.SecureMsgSend	chipZka	Command APDU PUT DATA	fill RND1
DK.SecureMsgReceive	chipZka	Response APDU	
DK.SecureMsgSend	chipZka	Command APDU READ RECORD EFINFO with Secure Messaging	
DK.SecureMsgReceive	chipZka	record EFINFO	check MAC
DK.SecureMsgSend	chipZka	Command APDU GET CHALLENGE	

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Command	protocol	msg	Service Provider's actions
DK.SecureMsgReceive	chipZka	Random number RND2 from chip	store RND2
DK.SecureMsgSend	chipZka	Command APDU VERIFY	provide complete command APDU
DK.SecureMsgReceive	chipZka	Response APDU	
PIN Verification Type 1			
DK.SecureMsgSend	chipZka	Command APDU GET KEYINFO	
DK.SecureMsgReceive	chipZka	Response APDU	
DK.SecureMsgSend	chipZka	Command APDU GET CHALLENGE	
DK.SecureMsgReceive	chipZka	Random number RND0 from chip	store RND0
DK.SecureMsgSend	chipZka	Command APDU MUTUAL AUTHENTICATE	fill ENC0
DK.SecureMsgReceive	chipZka	Response APDU	check ENC1
DK.SecureMsgSend	chipZka	Command APDU VERIFY	provide complete command APDU
DK.SecureMsgReceive	chipZka	Response APDU	check MAC
"Laden vom Kartenkonto" (both types)			
DK.SecureMsgSend	chipZka	Command APDU LADEN EINLEITEN	fill -Terminal ID - Trace No.
DK.SecureMsgReceive	chipZka	Response APDU	
DK.SecureMsgSend	isoLz	ISO8583 message 0200 Ladeanfrage	fill - Traceno. (BMP 11) - RNDMES (BMP 57) - MAC (BMP 64) check other security relevant fields.
DK.SecureMsgReceive	isoLz	ISO8583 message 0210 Ladeantwort	check MAC and other security relevant fields.
DK.SecureMsgSend	chipZka	Command APDU LADEN	
DK.SecureMsgReceive	chipZka	Response APDU	
DK.GetJournal	isoLz	Vendor specific	
Reversal of a "Laden vom Kartenkonto"			
DK.SecureMsgSend	chipZka	Command APDU SELECT FILE DFBÖRSE	
DK.SecureMsgReceive	chipZka	Response APDU	

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Command	protocol	msg	Service Provider's actions
DK.SecureMsgSend	chipZka	Command APDU LADEN EINLEITEN	fill -Terminal ID - Traceno.
DK.SecureMsgReceive	chipZka	Response APDU	
DK.SecureMsgSend	isoLz	ISO8583 message 0400 Storno	fill - Traceno. (BMP 11) - RNDMES (BMP 57) - MAC (BMP 64) check other security relevant fields.
DK.SecureMsgReceive	isoLz	ISO8583 message 0410 Storno Response	check MAC and other security relevant fields
DK.SecureMsgSend	chipZka	Command APDU LADEN	
DK.SecureMsgReceive	chipZka	Response APDU	
DK.GetJournal	isoLz	Vendor specific	
unload			
DK.SecureMsgSend	chipZka	ENTLADEN EINLEITEN	fill -Terminal ID - Trace No.
DK.SecureMsgReceive	chipZka	Response APDU	
DK.SecureMsgSend	isoLz	ISO8583 message Entladeanfrage 0200	fill - Traceno. (BMP 11) - RNDMES (BMP 57) - MAC (BMP 64) check other security relevant fields.
DK.SecureMsgReceive	isoLz	ISO8583 message Entladeantwort 0210	check MAC and other security relevant fields
DK.SecureMsgSend	chipZka	ENTLADEN	
DK.SecureMsgReceive	chipZka	Response APDU	
DK.SecureMsgSend	chipZka	ENTLADEN EINLEITEN	fill -Terminal ID - Trace No.
DK.SecureMsgReceive	chipZka	Response APDU	
DK.SecureMsgSend	isoLz	ISO8583 message Entladequittung 0202	fill - Traceno. (BMP 11) - RNDMES (BMP 57) - MAC (BMP 64) check other security relevant fields.
DK.SecureMsgReceive	isoLz	ISO8583 message Entladebestätigung 0212	check MAC and other security relevant fields
DK.SecureMsgSend	chipZka	Command APDU ENTLADEN	
DK.SecureMsgReceive	chipZka	Response APDU	

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Command	protocol	msg	Service Provider's actions
DK.GetJournal	isoLz	Vendor specific	
Repeated Messages (Stornowiederholung / Entladequittungswiederholung)			
DK.SecureMsgSend	isoLz	ISO8583 message Stornowiederholung 0401 or Entladequittungswiederholung 0203	fill - Traceno. (BMP 11) - RNDMES (BMP 57) - MAC (BMP 64) check other security relevant fields.
DK.SecureMsgReceive	isoLz	ISO8583 message Stornoantwort 410 or Entladebestätigung 0212	check MAC and other security relevant fields
DK.GetJournal	isoLz	Vendor specific	
Command	protocol	msg	Service Provider's actions
Preparation for PIN Change			
DK.SecureMsgSend	chipPinChg	Command APDU READ RECORD EFID	
DK.SecureMsgReceive	chipPinChg	Response APDU Record EFID Store EFID Will be inserted into BMP62 of a PIN Change request	
DK.SecureMsgSend	chipPinChg	Command APDU GET CHALLENGE	
DK.SecureMsgReceive	chipPinChg	Random number RND0 from Chip	Store RND0
DK.SecureMsgSend	chipPinChg	Command APDU READ RECORD EFINFO	Fill RND1
DK.SecureMsgReceive	chipPinChg	Response APDU Record EFINFO	Check MAC, Store EFINO Will be inserted into BMP62 of a PIN Change request
DK.SecureMsgSend	chipPinChg	Command APDU GET KEYINFO	
DK.SecureMsgReceive	chipPinChg	Response APDU Version of KCard	Store version byte Will be inserted into BMP62 of a PIN Change request
DK.SecureMsgSend	chipPinChg	Command APDU SEARCH RECORD '01' of EFPWDD	
DK.SecureMsgReceive	chipPinChg	Response APDU	Store record number Will be inserted into BMP62 of a PIN Change request

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Command	protocol	msg	Service Provider's actions
DK.SecureMsgSend	chipPinChg	Command APDU READ RECORD EFFBZ	
DK.SecureMsgReceive	chipPinChg	Response APDU Initial value FBZ Actual value FBZ	
PIN Verification			
DK.SecureMsgSend	chipPinChg	Command APDU GET KEYINFO	
DK.SecureMsgReceive	chipPinChg	Response APDU	
DK.SecureMsgSend	chipPinChg	Command APDU GET CHALLENGE	
DK.SecureMsgReceive	chipPinChg	Random number RND0 from chip	Store RND0
DK.SecureMsgSend	chipPinChg	Command APDU MUTUAL AUTHENTICATE	Fill ENC0
DK.SecureMsgReceive	chipPinChg	Response APDU	Check ENC1
DK.SecureMsgSend	chipPinChg	Command APDU VERIFY	Provide complete command APDU
DK.SecureMsgReceive	chipPinChg	Response APDU	Check MAC Create PAC for old PIN
PIN Change			
Let the user enter the PIN for the first time, by invoking the command Keyboard.PinEntry			
DK.SecureMsgSend	HSMPinCmp	Byte 0: 0x01 (Save PIN)	Save the PIN value entered for subsequent compare. Output data buffer length is zero.
Let the user enter the PIN for the second time, by invoking the command Keyboard.PinEntry			
DK.SecureMsgSend	HSMPinCmp	Byte 0: 0x02 (Compare PINs)	Compare PIN values. Returns Byte 0: as 0x00 when PIN does not match, and 0x01 when PIN does match. Create PAC for new PIN if values match
DK.SecureMsgSend	chipPinChg	Command APDU MANAGE SECURITY ENVIRONMENT	
DK.SecureMsgReceive	chipPinChg	Response APDU	
DK.SecureMsgSend	chipPinChg	Command APDU GET CHALLENGE	

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Command	protocol	msg	Service Provider's actions
DK.SecureMsgReceive	chipPinChg	Random number RND0 from Chip	Store RND0Will be inserted into BMP62 of a PIN Change request
DK.SecureMsgSend	isoPinChg	ISO8583 Message 0640	Fill - PAC old PIN (BMP52) - KTerminal generation + KTerminal version + RNDMES + RNDPAC (BMP57) - Chip Data (BMP62) with PAC of new PIN - MAC (BMP64)
DK.SecureMsgReceive	isoPinChg	ISO8583 message 0650	Check MAC
DK.SecureMsgSend	chipPinChg	Command APDU from BMP62	
DK.SecureMsgReceive	chipPinChg	Response APDU	
PIN Change Confirmation/ Repeated Confirmation			
DK.SecureMsgSend	isoPinChg	ISO8583 message 0642 or 0643 BMP25 = 00	Fill - KTerminal generation + KTerminal version + RNDMES (BMP57) - Chip Data (BMP62) with PAC of new PIN - MAC (BMP64)
DK.SecureMsgReceive	isoPinChg	ISO8583 message 0652	Check MAC
PIN Change Reversal/ Repeated Reversal			
DK.SecureMsgSend	isoPinChg	ISO8583 message 0642 or 0643 BMP25 ≠ 00	Fill - KTerminal generation + KTerminal version + RNDMES (BMP57) - Chip Data (BMP62) with PAC of old PIN - MAC (BMP64)
DK.SecureMsgReceive	isoPinChg	ISO8583 message 0652	Check MAC
PIN Activation after failure			

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Command	protocol	msg	Service Provider's actions
DK.SecureMsgSend	isoPinChg	ISO8583 message 0640	Fill - PAC entered PIN (BMP52) - KTerminal generation + KTerminal version + RNDMES + RNDPAC (BMP57) - Chip Data (BMP62) with PAC of entered PIN - MAC (BMP64)
DK.SecureMsgReceive	isoPinChg	ISO8583 message 0650	Check MAC
PIN Activation			
DK.SecureMsgSend	chipPinChg	Command APDU MANAGE SECURITY ENVIRONMENT	
DK.SecureMsgReceive	chipPinChg	Response APDU	
DK.SecureMsgSend	chipPinChg	Command APDU GET CHALLENGE	
DK.SecureMsgReceive	chipPinChg	Random number RND0 from Chip	Store RND0 Will be inserted into BMP62 of a PIN Activation request
DK.SecureMsgSend	isopinchg	ISO8583 Message 0640	Fill - PAC entered PIN (BMP52) - KTerminal generation + KTerminal version + RNDMES + RNDPAC (BMP57) - Chip Data (BMP62) with PAC of entered PIN - MAC (BMP64)
DK.SecureMsgReceive	isopinchg	ISO8583 message 0650	Check MAC
DK.SecureMsgSend	chipPinChg	Command APDU from BMP62	
DK.SecureMsgReceive	chipPinChg	Response APDU	
PIN Activation Confirmation/ Repeated Confirmation			
DK.SecureMsgSend	chipPinChg	Command APDU MANAGE SECURITY ENVIRONMENT	
DK.SecureMsgReceive	chipPinChg	Response APDU	
DK.SecureMsgSend	chipPinChg	Command APDU GET CHALLENGE	

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Command	protocol	msg	Service Provider's actions
DK.SecureMsgReceive	chipPinChg	Random number RND0 from Chip	Store RND0 Will be inserted into BMP62 of a PIN Activation confirmation
DK.SecureMsgSend	isopinchg	ISO8583 message 0642 or 0643 BMP25 = 00	Fill - KTerminal generation + KTerminal version + RNDMES (BMP57) - Chip Data (BMP62) with PAC of entered PIN - MAC (BMP64)
DK.SecureMsgReceive	isopinchg	ISO8583 message 0652	Check MAC
DK.SecureMsgSend	chipPinChg	Command APDU from BMP62	
DK.SecureMsgReceive	chipPinChg	Response APDU	

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

18.3 Command Messages

18.3.1 DK.GetHSMTData

This function returns the current HSM terminal data. The data is returned as a series of 'tag/length/value' items.

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 canceled.
default: 0

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"tData": Add example to YAML`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

tData
Contains the parameter settings as a series of 'tag/length/value' items with no separators. See command DK.hsmSetTData for the tags supported. The Base64 encoded data.

Event Messages

None

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

18.3.2 DK.GetQueryLogicalHSMDetail

This command reports the ZKA logical hsm available within the EPP. It also reports which logical HSM is currently active.

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 canceled.
default: 0

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"activeLogicalHSM": 0,`	integer	
`"hsmInfo": [{`	array (object)	
`"hsmSerialNumber": 0,`	integer	
`"zkaid": Add example to YAML`	string	
`}]`		
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

activeLogicalHSM
Specifies the serial number of the logical hsm that is currently active. This value is the hsm serial number (tag CB in the hsm TDATA) encoded as a normal binary value (i.e. it is not a bcd).

hsmInfo
array of hsmInfo(one for each logical HSM).

hsmInfo/
hsm information

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
hsmInfo/hsmSerialNumber Specifies the Serial Number of the Logical HSM (tag CB in the hsm tData). This value is encoded as a normal binary value (i.e. it is not a BCD).
hsmInfo/zkaid A string containing the ZKA ID of the logical HSM (defined by tag CC in the hsm tData). The characters in the string are EBCIDIC characters

Event Messages

None

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

18.3.3 DK.HSMSetTData

This function allows the application to set the HSM terminal data (except keys, trace number and session key index). The data must be provided as a series of 'tag/length/value' items. Terminal data that are set but are not supported by the hardware will be ignored.

Command Message

Payload	Type	Required
`{`		
`  "timeout": 5000,`	integer	
`  "tData": 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 canceled.

default: 0

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties

tData

Specifies which parameter(s) is(are) to be set. The tData is a series of "tag/length/value" items where each item consists of:

- One byte tag (see the list of tags below).
- One byte specifying the length of the following data as an unsigned binary number.
- N bytes data (see the list below for formatting) with no separators. The following tags are supported:

Tag (hexl)	Format	Length (bytes)	Meaning	Read/Write	EPP/HSM
C2	BCD	4	Terminal ID. ISO BMP 41	R/W	EPP
C3	BCD	4	Bank code. ISO BMP 42 (rightmost 4 bytes) Account data for terminal account	R/W	EPP
C4	BCD	9	Account data for terminal account. ISO BMP 60 (load against other card)	R/W	EPP
C5	BCD	9	Account data for fee account. ISO BMP 60 (Laden vom Kartenkonto)	R/W	EPP
C6	EBCDIC	40	Terminal location. ISO BMP 43	R/W	EPP
C7	ASCII	3	Terminal currency	R/W	EPP
C8	BCD	7	Online date and time. ISO BMP 61(YYYYMMDDHHMMSS)	R/W	HSM
C9	BCD	4	Minimum load fee in units of 1/100 of terminal currency, checked against leftmost 4 Bytes of ISO BMP42	R/W	EPP
CA	BCD	4	Maximum load fee in units of 1/100 of terminal currency, checked against leftmost 4 Bytes of ISO BMP42	R/W	EPP
CB	BIN	3	logical HSM binary coded serial number (starts with 1; 0 means that there are no logical HSMs)	R	HSM
CC	EBCDIC	16	ZKA ID (is filled during the pre-initialization of the HSM)	R	HSM
CD	BIN	1	HSM status	R	HSM
			1 = irreversibly out of order		
			2 = out of order, K_UR is not loaded		
			3 = not pre-initialized, K_UR is loaded		
			4 = pre-initialized, K_INIT is loaded		
			5 = initialized/personalized, K_PERS is loaded		
CE	EBCDIC	variable	HSM-ID (6 byte Manufacturer- ID + min. 16 min. 10 Byte serial number) as needed for ISO BMP57 of a pre-initialization	R	EPP

In the table above, the fifth column indicates if the variable is read only or both read and write. The sixth column indicates if the variable is unique per logical HSM or common across all logical HSMs within an EPP. The Base64 encoded data.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "accessDenied"`	string	
`}`		

Properties

completionCode

The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription

If included, this contains additional vendor dependent information to assist with problem resolution.

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.
- `hsmStateInvalid` - The HSM is not in a correct state to handle this command.

Event Messages

None

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

18.3.4 DK.SecureMsgSend

This function allows the application to set the HSM terminal data (except keys, trace number and session key index). The data must be provided as a series of 'tag/length/value' items. Terminal data that are set but are not supported by the hardware will be ignored.

Command Message

Payload	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"protocol": "isoAs",`	string	
`"msg": 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 canceled.
default: 0

protocol
Specifies the protocol the message belongs to. The following values are possible:

- `isoAs` - ISO 8583 protocol for the authorization system.
- `isoLz` - ISO 8583 protocol for the German Ladezentrale.
- `isoPs` - ISO 8583 protocol for the personalization system.
- `chipZka` - ZKA chip protocol.
- `rawData` - Raw data protocol.
- `pbm` - PBM protocol.(see [Ref. dk-8] – [Ref. dk-13]).
- `hsmLdi` - HSM LDI protocol.
- `genAs` - Generic PAC/MAC for non-ISO8583 message formats.
- `chipPinChg` - ZKA chip protocol for changing the PIN on a GeldKarte.
- `pinCmp` - Protocol for comparing PIN numbers entered in the PIN pad during a PIN Change transaction.
- `isoPinChg` - ISO8583 authorization system protocol for changing the PIN on a GeldKarte.

msg
Specifies the message that was received. This property is omitted if during a specified time period no response was received from the communication partner (necessary to set the internal state machine to the correct state). The Base64 encoded data.

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "accessDenied",`	string	
`"protocol": "isoAs",`	string	

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Payload	Type	Required
`"msg": Add example to YAML`	string	
`}`		

Properties

completionCode
The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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.
- `hsmStateInvalid` - The HSM is not in a correct state to return journal data
- `protocolInvalid` - The specified protocol is invalid.
- `formatInvalid` - The format of the message is invalid.
- `contentInvalid` - The contents of one of the security relevant properties are invalid.
- `keyNotFound` - No key was found for PAC/MAC generation.
- `noPin` - No PIN or insufficient PIN-digits have been entered.

protocol
Specifies the protocol the message belongs to. The following values are possible:

- `isoAs` - ISO 8583 protocol for the authorization system.
- `isoLz` - ISO 8583 protocol for the German Ladezentrale.
- `isoPs` - ISO 8583 protocol for the personalization system.
- `chipZka` - ZKA chip protocol.
- `rawData` - Raw data protocol.
- `pbm` - PBM protocol.(see [Ref. dk-8] – [Ref. dk-13]).
- `hsmLdi` - HSM LDI protocol.
- `genAs` - Generic PAC/MAC for non-ISO8583 message formats.
- `chipPinChg` - ZKA chip protocol for changing the PIN on a GeldKarte.
- `pinCmp` - Protocol for comparing PIN numbers entered in the PIN pad during a PIN Change transaction.
- `isoPinChg` - ISO8583 authorization system protocol for changing the PIN on a GeldKarte.

msg
Specifies the message that was received. This property is omitted if during a specified time period no response was received from the communication partner (necessary to set the internal state machine to the correct state). The Base64 encoded data.

Event Messages

None

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

18.3.5 DK.SecureMsgReceive

This command handles all messages that are received through a secure messaging from an authorization system, German 'Ladezentrale', personalization system or the chip. The encryption module checks the security relevant properties. All messages must be presented to the encryptor via this command even if they do not contain security relevant properties in order to keep track of the transaction status in the internal state machine.

Command Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "accessDenied",`	string	
`"protocol": "isoAs",`	string	
`"msg": Add example to YAML`	string	
`}`		

Properties

completionCode
The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

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.
- `hsmStateInvalid` - The HSM is not in a correct state to return journal data
- `macInvalid` - The MAC of the message is not correct.
- `protocolInvalid` - The specified protocol is invalid.
- `formatInvalid` - The format of the message is invalid.
- `contentInvalid` - The contents of one of the security relevant properties are invalid.
- `keyNotFound` - No key was found for PAC/MAC generation.

protocol
Specifies the protocol the message belongs to. The following values are possible:

- `isoAs` - ISO 8583 protocol for the authorization system.
- `isoLz` - ISO 8583 protocol for the German Ladezentrale.
- `isoPs` - ISO 8583 protocol for the personalization system.
- `chipZka` - ZKA chip protocol.
- `rawData` - Raw data protocol.
- `pbm` - PBM protocol.(see [Ref. dk-8] – [Ref. dk-13]).
- `hsmLdi` - HSM LDI protocol.
- `genAs` - Generic PAC/MAC for non-ISO8583 message formats.
- `chipPinChg` - ZKA chip protocol for changing the PIN on a GeldKarte.
- `pinCmp` - Protocol for comparing PIN numbers entered in the PIN pad during a PIN Change transaction.
- `isoPinChg` - ISO8583 authorization system protocol for changing the PIN on a GeldKarte.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
msg Specifies the message that was received. This field is omitted if during a specified time period no response was received from the communication partner (necessary to set the internal state machine to the correct state). The Base64 encoded data.

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	string	
`}`		

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

18.3.6 DK.GetJournal

This command is used to get journal data from the encryption module. It retrieves cryptographically secured information about the result of the last transaction that was done with the indicated protocol. When the Service Provider supports journaling (see hsmjournaling) then it is impossible to do any DK.SecureMsgSend/DK.SecureMsgReceive with this protocol, unless the journal data is retrieved. It is possible - especially after restarting a system - to get the same journal data again.

Command Message

Payload	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"protocol": "isoas"`	string	
`}`		

Properties

timeout
Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be canceled.
default: 0

protocol
Specifies the protocol the journal data belong to. The following values are possible:

- `isoas` Get authorization system journal data.
- `isolz` Get German Ladezentrale journal data.
- `isops` Get personalization system journal data.
- `pbm` Get PBM protocol data.

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "accessDenied",`	string	
`"journalData": Add example to YAML`	string	
`}`		

Properties

completionCode
The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
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. • `hsmStateInvalid` The HSM is not in a correct state to return journal data • `protocolInvalid` The specified protocol is invalid.
journalData The journal data encoded in base64.

Event Messages

None

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

18.3.7 DK.HSMInit

This command is used to set the hsm out of order. If multiple logical hsms are configured then the command sets the currently active logical hsm out of order. At the same time the online time can be set to control when the opt online dialog (see Wisops protocol) shall be started to initialize the hsm again. When this time is reached an DK.OPTRequiredEvent will be sent.

Command Message

Payload	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"initMode": "temp",`	string	
`"onlineTime": 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 canceled.

default: 0

initMode

Specifies the init mode. The following values are possible:

- `temp` - Initialize the HSM temporarily (K_UR remains loaded).
- `definite` - Initialize the HSM definitely (K_UR is deleted).
- `irreversible` - Initialize the HSM irreversibly (can only be restored by the vendor).

onlineTime

Specifies the Online date and time in the format YYYYMMDDHHMMSS like in ISO BMP 61 as BCD packed characters. This parameter is omitted when the init mode equals definite or irreversible.

Property value constraints:

```
pattern: ^[0-9]{4}(0[1-9]|1[0-2])(0[1-9]|[12][0-9]|3[01])([01][0-9]|2[0-3])[0-5][0-9][0-5][0-9]$
```

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "modeNotSupported"`	string	
`}`		

Properties

completionCode

The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
errorCode Specifies the error code if applicable. The following values are possible: • `modeNotSupported` - The specified init mode is not supported. • `hsmStateInvalid` - The HSM is not in a correct state to handle this command.

Event Messages

None

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

18.3.8 DK.SetLogicalHSM

This command allows an application select the logical HSM that should be active. If the device does not support multiple logical hsm this command returns Unsupported. The DK.GetQueryLogicalHSMDetail command can be called to determine the current active logical HSM.

Once the active logical HSM is set with this command, that logical hsm remains active until this command is used to change the logical hsm or the system is re-started.

The selected HSM is not persistent across re-boots, when applications want to address a specific logical HSM they must ensure that the correct logical hsm is set as the active logical hsm.

The commands affected by this command are as follows:

- DK.GetHSMTData
- KeyManagement.KeyDetail
- DK.HSMSetTData
- DK.SecureMsgSend (only affected for the protocols hsmldi and isops)
- DK.SecureMsgReceive (only affected for the protocols hsmldi and isops)
- DK.HSMInit
- DK.GetJournal (only affected for the protocol isops).

If there are multiple applications that manipulate the current logical hsm then applications must co-operate or use the XFS locking facilities to synchronize access to the logical hsm. The current logical hsm is the same for all clients.

Command Message

Payload	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"hsmSerialNumber": 0`	integer	
`}`		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be canceled.

default: 0

hsmSerialNumber

Specifies the serial number of the hsm that should be set as the active hsm. The value passed in this property corresponds to the hsmSerialNumber property reported in the DK.GetQueryLogicalHSMDetail command output (and hence corresponds to the CB tag in the hsm tData). The hsmSerialNumber value is encoded as a standard binary value

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "accessDenied"`	string	

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Payload	Type	Required
`}`		

Properties
completionCode The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
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. • `invalidHsm` - The logical HSM serial number specified is not valid.

Event Messages

None

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

18.4 Unsolicited Messages

18.4.1 DK.HSMTDataChangedEvent

This event indicates that one of the values of the terminal data has changed (these are the data that can be set using HSMSetTData). I.e. this event will be sent especially when the online time or the hsm status is changed because of a DK.HSMInit command or an OPT online dialog (DK.SecureMsgSend/DK.SecureMsgReceive with PROTISOPS).

On configurations with multiple logical HSMs, the serial number tag must be included within the data so that the logical HSM that has changed can be identified.

Payload	Type	Required
`{`		
`"tData": Add example to YAML`	string	Yes
`}`		

Properties
tData Contains the parameter settings as a series of 'tag/length/value' items. See command DK.HSMSetTData for the tags supported. The Base64 encoded data.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

18.4.2 DK.HSMChangedEvent

This event indicates that the currently active logical HSM has been changed. This event will be triggered when an application changes the current HSM through the DK.GetLogicalHSM command. This event is not generated if the HSM is not changed.

Payload	Type	Required
`{`		
`"hsmSerialNumber": 0`	integer	Yes
`}`		

Properties

hsmSerialNumber

Specifies the serial number of the logical hsm that has been made active. The hsmSerialNumber value is encoded as a standard binary value (i.e. it is not BCD).

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

18.4.3 DK.OPTRequiredEvent

This event indicates that the online date/time stored in a HSM has been reached.

Payload	Type	Required
`{`		
`"hsmSerialNumber": 0`	integer	
`}`		

Properties

hsmSerialNumber

Specifies the serial number of the logical hsm where the online time has been reached. If logical hsms are not supported then this property is omitted. The hsmSerialNumber value is encoded as a standard binary value. (i.e. it is not BCD).

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

19. Lights Interface

This chapter defines the Lights interface functionality and messages.

19.1 Summary

This specification describes the functionality of the services provided by the Lights service by defining the service-specific commands that can be issued. This service allows for the operation of Lights, LEDs and Lamps on a device.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

19.2 Command Messages

19.2.1 Lights.SetLight

This command is used to set the status of a light.

For guidelights, the slow and medium flash rates must not be greater than 2.0 Hz. It should be noted that in order to comply with American Disabilities Act guidelines only a slow or medium flash rate must be used.

Command Message

Payload	Type	Required
`{`		
`  "timeout": 5000,`	integer	
`  "cardReader": {`	object	
`    "position": "left",`	string	
`    "flashRate": "off",`	string	
`    "colour": "red",`	string	
`    "direction": "entry"`	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.`		
`  },`		

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Payload	Type	Required
"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.		
},		
"statusGood": {	object	
See cardReader properties.		
},		
"statusWarning": {	object	
See cardReader properties.		
},		
"statusBad": {	object	
See cardReader properties.		
},		
"statusSupervisor": {	object	

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Payload	Type	Required
`See cardReader properties.`		
`},`		
`"statusInService": {`	object	
`See cardReader properties.`		
`},`		
`"fasciaLight": {`	object	
`See cardReader properties.`		
`},`		
`"exampleProperty1": {`	object	
`See cardReader properties.`		
`},`		
`"exampleProperty2": {`	object	
`See exampleProperty1 properties.`		
`}`		
`}`		

Properties

timeout
Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be canceled.
default: 0

cardReader
Card Unit Light.

cardReader/position
The light position. If omitted then the default position is used. One of the following values:

- `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.

cardReader/flashRate
The light flash rate as one of the following values:

- `off` - The light is turned off.
- `slow` - The light is flashing slowly.
- `medium` - The light is flashing medium frequency.
- `quick` - The light is flashing quickly.
- `continuous` - The light is continuous (steady).

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
cardReader/colour The light color as one of the following values: • `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.
cardReader/direction The light direction as one of the following values: • `entry` - The light is indicating entry. • `exit` - The light is indicating exit.
pinPad Pin Pad Light.
notesDispenser Notes Dispenser Light.
coinDispenser Coin Dispenser Light.
receiptPrinter Receipt Printer Light.
passbookPrinter Passbook Printer Light.
envelopeDepository Envelope Depository Light.
chequeUnit Cheque Unit Light.
billAcceptor Bill Acceptor Light.
envelopeDispenser Envelope Dispenser Light.
documentPrinter Document Printer Light.
coinAcceptor Coin Acceptor Light.
scanner scanner Light.
contactless Contactless Light.
cardUnit2 Card Unit 2 Light.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
notesDispenser2 Notes Dispenser 2 Light.
billAcceptor2 Bill Acceptor 2 Light.
statusGood Status indicator light - Good.
statusWarning Status indicator light - Warning.
statusBad Status indicator light - Bad.
statusSupervisor Status indicator light - Supervisor.
statusInService Status indicator light - In Service.
fasciaLight Fascia Light.
exampleProperty1 (example name)

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "invalidAuxiliary"`	string	
`}`		

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
errorCode Specifies the error code if applicable. The following values are possible: • `invalidAuxiliary` - An attempt to set an auxiliary to a new value was invalid because the auxiliary does not exist or the auxiliary is pre-configured as an input port. • `AuxiliaryError` - A hardware error occurred while executing the command. • `syntaxError` - The command was invoked with incorrect input data.

Event Messages

None

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

20. Auxiliaries Interface

This chapter defines the Auxiliaries interface functionality and messages.

20.1 Summary

This specification describes the functionality of the services provided by the Auxiliaries service by defining the service-specific commands that can be issued. This section describes the functions provided by a generic Auxiliaries Unit service. This service allows for the operation of the following categories of Auxiliaries:

- Door sensors, such as cabinet, safe or vandal shield doors.
- Alarm sensors, such as tamper, seismic or heat sensors.
- Generic sensors, such as proximity or ambient light sensors.
- Key switch sensors, such as the ATM operator switch.
- Lamp/sign indicators, such as fascia light or audio indicators.
- Auxiliary indicators.
- Enhanced Audio Controller, for use by the partially sighted.

In self-service devices, the Auxiliaries unit is capable of dealing with external sensors, such as door switches, locks, alarms and proximity sensors, as well as external indicators, such as turning on lamps or heating.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

20.2 Command Messages

20.2.1 Auxiliaries.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 canceled.
default: 0

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"mode": "clear",`	string	
`"startTime": {`	object	
`"year": 1601,`	integer	
`"month": 1,`	integer	
`"dayOfWeek": "Saturday",`	string	
`"day": 1,`	integer	
`"hour": 0,`	integer	
`"minute": 0`	integer	
`}`		
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
mode Specifies the current auto start-up control mode configured. The following values are possible: • `clear` - No auto start-up time configured. • `specific` - In the *startTime* object, only *year*, *month, *day*, *hour* and *minute* are relevant. All other field must be ignored. • `daily` - Auto start-up every day has been configured. In the *startTime* object, only *hour* and *minute* are relevant. All other properties must be ignored. • `weekly` - Auto start-up at a specified time on a specific day of every week has been configured. In the *startTime* parameter, only *dayOfWeek*, *hour* and *minute* are relevant. All other properties must be ignored.
startTime Specifies the current auto start-up time configuration. For reference see the Win32 SYSTEMTIME structure.

startTime/year
Specifies the year. The value should be ignored if it is not relevant to the mode value.
Property value constraints:
`minimum: 1601`
`maximum: 30827`

startTime/month
Specifies the month. The value should be ignored if it is not relevant to the mode value.
Property value constraints:
`minimum: 1`
`maximum: 12`

startTime/dayOfWeek Specifies the day of the week. The following values are possible: • `Saturday` - the day of the week is Saturday. • `Sunday` - the day of the week is Sunday. • `Monday` - the day of the week is Monday. • `Tuesday` - the day of the week is Tuesday. • `Wednesday` - the day of the week is Wednesday. • `Thursday` - the day of the week is Thursday. • `Friday` - the day of the week is Friday.

startTime/day
Specifies the day. The value should be ignored if it is not relevant to the mode value.
Property value constraints:
`minimum: 1`
`maximum: 31`

startTime/hour
Specifies the hour. The value should be ignored if it is not relevant to the mode value.
Property value constraints:
`minimum: 0`
`maximum: 23`

startTime/minute
Specifies the minute. The value should be ignored if it is not relevant to the mode value.
Property value constraints:
`minimum: 0`
`maximum: 59`

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Event Messages

None

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

20.2.2 Auxiliaries.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 canceled.
default: 0

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

20.2.3 Auxiliaries.Register

This command is used to register for, or deregister events from the Auxiliaries 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 events will be generated which indicates the auxiliaries 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	
"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	
"volumeControl": "register",	string	

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Payload	Type	Required
`"ups": "register",`	string	
`"remoteStatusMonitor": "register",`	string	
`"audibleAlarm": "register",`	string	
`"enhancedAudioControl": "register",`	string	
`"enhancedMicrophoneControl": "register",`	string	
`"microphoneVolume": "register",`	string	
`"exampleProperty1": "register",`	string	
`"exampleProperty2": "register"`	string	
`}`		

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.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
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.
headsetMicrophone "pecifies 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.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
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.
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.
exampleProperty1 (example name) Specifies whether the vendor dependent sensors should report whenever they change status. See operatorSwitch for the possible values.

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "invalidAuxiliary"`	string	
`}`		

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
errorCode Specifies the error code if applicable. The following values are possible: • `invalidAuxiliary` - An attempt to register for or disable events to a auxiliary was invalid because the auxiliary 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 auxiliary was made.

Event Messages

None

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

20.2.4 Auxiliaries.SetAuxiliaries

This command is used to set or clear one or more device auxiliaries.

Command Message

Payload	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"safeDoor": "bolt",`	string	
`"vandalShield": "closed",`	string	
`"frontCabinetDoors": "bolt",`	string	
`"rearCabinetDoors": "bolt",`	string	
`"leftCabinetDoors": "bolt",`	string	
`"rightCabinetDoors": "bolt",`	string	
`"openClose": "closed",`	string	
`"fasciaLight": "off",`	string	
`"audio": {`	object	
`"rate": "on",`	string	
`"signal": "keypress"`	string	
`},`		
`"heating": "on",`	string	
`"displayBackLight": "on",`	string	
`"signageDisplay": "on",`	string	
`"volume": 1,`	integer	
`"ups": "engage",`	string	
`"audibleAlarm": "off",`	string	
`"enhancedAudioControl": "publicAudioManual",`	string	
`"enhancedMicrophoneControl": "publicAudioManual",`	string	
`"microphoneVolume": 1`	integer	
`}`		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be canceled.

default: 0

safeDoor

Specifies whether the safe doors should be bolted or unbolted as one of the following values:

- `bolt` - All Safe Doors are bolted.
- `unbolt` - All Safe Doors are unbolted.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
vandalShield Specifies whether the Vandal Shield should change position as one of the following values: • `closed` - Close the Vandal Shield. • `open` - Open the Vandal Shield. • `service` - Position the Vandal Shield in the service position. • `keyboard` - Position the Vandal Shield to permit access to the keyboard.
frontCabinetDoors Specifies whether all the front Cabinet Doors should be bolted or unbolted as one of the following values: • `bolt` - All front Cabinet Doors are bolted. • `unbolt` - All front Cabinet Doors are unbolted.
rearCabinetDoors Specifies whether all the rear Cabinet Doors should be bolted or unbolted as one of the following values: • `bolt` - All rear Cabinet Doors are bolted. • `unbolt` - All rear Cabinet Doors are unbolted.
leftCabinetDoors Specifies whether allthe left Cabinet Doors should be bolted or unbolted as one of the following values: • `bolt` - All left Cabinet Doors are bolted. • `unbolt` - All left Cabinet Doors are unbolted.
rightCabinetDoors Specifies whether all the right Cabinet Doors should be bolted or unbolted as one of the following values: • `bolt` - All right Cabinet Doors are bolted. • `unbolt` - All right Cabinet Doors are unbolted.
openClose Specifies whether the Open/Closed Indicator should show Open or Close to a consumer as one of the following values: • `closed` - The Open/Closed Indicator is changed to show that the terminal is closed for a consumer. • `open` - The Open/Closed Indicator is changed to show that the terminal is open to be used by a consumer.
fasciaLight Specifies whether the Fascia Lights should be turned on or off as one of the following values: • `off` - Turn off the Fascia Light. • `on` - Turn on the Fascia Light.
audio Specifies whether the Audio Indicator should be turned on or off, if available.
audio/rate Specifies the rate of the Audio Indicator as one of the following values: • `on` - Turn on the Audio Indicator. • `off` - Turn off the Audio Indicator. • `continuous` - Turn the Audio Indicator to continuous.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
audio/signal Specifies the Audio sound as one of the following values: • `keypress` - Sound a key click signal. • `exclamation` - Sound an exclamation signal. • `warning` - Sound a warning signal. • `error` - Sound an error signal. • `critical` - Sound a critical error signal.
heating Specifies whether the Internal Heating device should be turned on or off as one of the following values: • `off` - The Internal Heating device is turned off. • `on` - The Internal Heating device is turned on.
displayBackLight Specifies whether the Consumer Display Backlight should be turned on or off as one of the following values: • `off` - The Consumer Display Backlight is turned off. • `on` - The Consumer Display Backlight is turned on.
signageDisplay Specifies whether the Signage Display should be turned on or off as one of the following values: • `off` - The Signage Display is turned off. • `on` - The Signage Display is turned on.

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.
Property value constraints:
`minimum: 1`
`maximum: 1000`

ups Specifies whether the UPS device should be engaged or disengaged. The UPS device should not be engaged when the charge level is low. Specified as one of the following values: • `engage` - Engage the UPS. • `disengage` - Disengage the UPS.
audibleAlarm Specifies whether the state of the Audible Alarm device should be changed as one of the following values: • `off` - Turn off the Audible Alarm device. • `on` - Turn on the Audible Alarm device.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties

enhancedAudioControl

Specifies whether the state of the Enhanced Audio Controller should be changed as one of the following values:

- `publicAudioManual` - Set the Enhanced Audio Controller to manual mode, public state (i.e. audio will be played through speakers only).
- `publicAudioAuto` - Set the Enhanced Audio Controller to auto mode, public state (i.e. audio will be played through speakers). When a Privacy Device is activated (headset connected/handset off-hook), the device will go to the private state.
- `publicAudioSemiAuto` - Set the Enhanced Audio Controller to semi-auto mode, public state (i.e. audio will be played through speakers). When a Privacy Device is activated, the device will go to the private state.
- `privateAudioManual` - Set the Enhanced Audio Controller to manual mode, private state (i.e. audio will be played only through a connected Privacy Device). In private mode, no audio is transmitted through the speakers.
- `privateAudioAuto` - Set the Enhanced Audio Controller to auto mode, private state (i.e. audio will be played only through an activated Privacy Device). In private mode, no audio is transmitted through the speakers. When a Privacy Device is deactivated (headset disconnected/handset on-hook), the device will go to the public state.
- `privateAudioSemiAuto` - Set the Enhanced Audio Controller to semi-auto mode, private state (i.e. audio will be played only through an activated Privacy Device). In private mode, no audio is transmitted through the speakers. When a Privacy Device is deactivated, the device will remain in the private state.

enhancedMicrophoneControl

Specifies whether the state of the Enhanced Microphone Controller should be changed as one of the following values:

- `publicAudioManual` - Set the Enhanced Microphone Controller to manual mode, public state (i.e. only the microphone in the fascia is active).
- `publicAudioAuto` - Set the Enhanced Microphone Controller to auto mode, public state (i.e. only the microphone in the fascia is active). When a Privacy Device with a microphone is activated (headset connected/handset off-hook), the device will go to the private state.
- `publicAudioSemiAuto` - Set the Enhanced Microphone Controller to semi-auto mode, public state (i.e. only the microphone in the fascia is active). When a Privacy Device with a microphone is activated, the device will go to the private state.
- `privateAudioManual` - Set the Enhanced Microphone Controller to manual mode, private state (i.e. audio input will be only via a microphone in the Privacy Device). In private mode, no audio input is transmitted through the fascia microphone.
- `privateAudioAuto` - Set the Enhanced Microphone Controller to auto mode, private state (i.e. audio input will be only via a microphone in the Privacy Device). In private mode, no audio input is transmitted through the fascia microphone. When a Privacy Device with a microphone is deactivated (headset disconnected/handset on-hook), the device will go to the public state.
- `privateAudioSemiAuto` - Set the Enhanced Microphone Controller to semi-auto mode, private state (i.e. audio input will be only via a microphone in the Privacy Device). In private mode, no audio input is transmitted through the fascia microphone. When a Privacy Device with a microphone is deactivated, the device will remain in the private state.

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.

Property value constraints:

```
minimum: 1
maximum: 1000
```

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "invalidAuxiliary"`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

errorCode
Specifies the error code if applicable. The following values are possible:

- `invalidAuxiliary` - An attempt to set a auxiliary to a new value was invalid because the auxiliary does not exist or the auxiliary is pre-configured as an input port.
- `syntaxError` - The command was invoked with incorrect input data.

Event Messages

- Auxiliaries.AuxiliaryStatusEvent

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

20.2.5 Auxiliaries.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.

If a new start-up time is set by this command it will replace any previously set start-up time.

Before the auto start-up can take place the operating system must be shut down.

Command Message

Payload	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"mode": "clear",`	string	
`"startTime": {`	object	
`"year": 1601,`	integer	
`"month": 1,`	integer	
`"dayOfWeek": "Saturday",`	string	
`"day": 1,`	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 canceled.

default: 0

mode

Specifies the current auto start-up control mode configured. The following values are possible:

- `clear` - No auto start-up time configured.
- `specific` - In the *startTime* object, only *year*, *month, *day*, *hour* and *minute* are relevant. All other field must be ignored.
- `daily` - Auto start-up every day has been configured. In the *startTime* object, only *hour* and *minute* are relevant. All other properties must be ignored.
- `weekly` - Auto start-up at a specified time on a specific day of every week has been configured. In the *startTime* parameter, only *dayOfWeek*, *hour* and *minute* are relevant. All other properties must be ignored.

startTime

Specifies the current auto start-up time configuration. For reference see the Win32 SYSTEMTIME structure.

startTime/year

Specifies the year. The value should be ignored if it is not relevant to the mode value.

Property value constraints:

```
minimum: 1601
maximum: 30827
```

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties

startTime/month
Specifies the month. The value should be ignored if it is not relevant to the mode value.
Property value constraints:
`minimum: 1`
`maximum: 12`

startTime/dayOfWeek Specifies the day of the week. The following values are possible: • `Saturday` - the day of the week is Saturday. • `Sunday` - the day of the week is Sunday. • `Monday` - the day of the week is Monday. • `Tuesday` - the day of the week is Tuesday. • `Wednesday` - the day of the week is Wednesday. • `Thursday` - the day of the week is Thursday. • `Friday` - the day of the week is Friday.

startTime/day
Specifies the day. The value should be ignored if it is not relevant to the mode value.
Property value constraints:
`minimum: 1`
`maximum: 31`

startTime/hour
Specifies the hour. The value should be ignored if it is not relevant to the mode value.
Property value constraints:
`minimum: 0`
`maximum: 23`

startTime/minute
Specifies the minute. The value should be ignored if it is not relevant to the mode value.
Property value constraints:
`minimum: 0`
`maximum: 59`

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	string	
`}`		

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Event Messages

None

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

20.3 Unsolicited Messages

20.3.1 Auxiliaries.AuxiliaryStatusEvent

This event is used to specify that an auxiliary has changed its state, due to the result of a command or some external condition. Reporting of this event is controlled by the Auxiliaries.Register command. Event reporting of individual Auxiliaries is disabled by default.

Payload	Type	Required
{ "operatorSwitch": "notAvailable", "tamperSensor": "notAvailable", "intTamperSensor": "notAvailable", "seismicSensorState": "notAvailable", "heatSensorState": "notAvailable", "proximitySensorState": "notAvailable", "ambientLightSensorState": "notAvailable", "enhancedAudioSensorState": "notAvailable", "bootSwitchSensorState": "notAvailable", "displaySensorState": "notAvailable", "operatorCallButtonSensorState": "notAvailable", "handsetSensorState": "notAvailable", "headsetMicrophoneSensorState": "notAvailable", "FasciaMicrophoneSensorState": "notAvailable", "safeDoorState": "notAvailable", "vandalShieldState": "notAvailable", "cabinetFrontDoorState": "notAvailable", "cabinetRearDoorState": "notAvailable", "cabinetLeftDoorState": "notAvailable", "cabinetRightDoorState": "notAvailable", "openClosedIndicatorState": "notAvailable", "audioState": { "notAvailable": false, "off": false, "keypress": false, "exclamation": false, "warning": false, "error": false, "critical": false, "continuous": false	 string object boolean boolean boolean boolean boolean boolean boolean boolean	

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Payload	Type	Required
`},`		
`"heatingState": "notAvailable",`	string	
`"consumerDisplayBacklightState": "notAvailable",`	string	
`"signageDisplayState": "notAvailable",`	string	
`volumeState": {`	object	
`"available": false,`	boolean	
`"volumeLevel": 1`	integer	
`},`		
`"UPSState": {`	object	
`"notAvailable": false,`	boolean	
`"low": false,`	boolean	
`"engaged": false,`	boolean	
`"powering": false,`	boolean	
`"recovered": false`	boolean	
`},`		
`"audibleAlarmState": "notAvailable",`	string	
`"enhancedAudioControlState": "notAvailable",`	string	
`"enhancedMicrophoneControlState": "notAvailable",`	string	
`microphoneVolumeState": {`	object	
`"available": false,`	boolean	
`"volumeLevel": 1`	integer	
`}`		
`}`		

Properties

operatorSwitch

Specifies the state of the Operator switch.

- `notAvailable` - The status is not available.
- `run` - The switch is in run mode.
- `maintenance` - The switch is in maintenance mode.
- `supervisor` - TThe switch is in supervisor mode.

tamperSensor

Specifies the state of the Tamper sensor.

- `notAvailable` - The tamper sensor is not available.
- `off` - There is no indication of a tampering attempt.
- `on` - There has been a tampering attempt.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
intTamperSensor Specifies the state of the Internal Tamper Sensor for the internal alarm. This sensor indicates whether the internal alarm has been tampered with (such as a burglar attempt). Specified as one of the following: • `notAvailable` - The internal tamper sensor is not available. • `off` - There is no indication of a tampering attempt. • `on` - There has been a tampering attempt.
seismicSensorState Specifies the state of the Seismic Sensor. This sensor indicates whether the terminal has been shaken (e.g. burglar attempt or seismic activity). Specified as one of the following: • `notAvailable` - The status sensor is not available. • `off` - TThe seismic activity has not been high enough to trigger the sensor. • `on` - The seismic or other activity has triggered the sensor.
heatSensorState Specifies the state of the Heat Sensor. This sensor is triggered by excessive heat (fire) near the terminal. Specified as one of the following: • `notAvailable` - The status is not available. • `off` - The heat has not been high enough to trigger the sensor. • `on` - The heat has been high enough to trigger the sensor.
proximitySensorState Specifies the state of the Proximity Sensor. This sensor is triggered by movements around the terminal. Specified as one of the following: • `notAvailable` - The status is not available. • `present` - The sensor is showing that there is someone present at the terminal. • `notPresent` - The sensor can not sense any people around the terminal.
ambientLightSensorState Specifies the state of the Ambient Light Sensor. This sensor indicates the level of ambient light around the terminal. Interpretation of this value is vendor-specific and therefore it is not guaranteed to report a consistent actual ambient light level across different vendor hardware. Specified as one of the following: • `notAvailable` - The status is not available. • `veryDark` - The level of light is very dark. • `dark` - The level of light is dark. • `mediumLight` - The level of light is medium light. • `light` - The level of light is light. • `veryLight` - The level of light is very light.
enhancedAudioSensorState Specifies the presence or absence of a consumer's headphone connected to the Audio Jack. Specified as one of the following: • `notAvailable` - The status is not available. • `present` - There is a headset connected. • `notPresent` - There is no headset connected.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
bootSwitchSensorState Specifies the state of the Boot Switch Sensor. This sensor is triggered whenever the terminal is about to be rebooted or shutdown due to a delayed effect switch. Specified as one of the following: • `notAvailable` - The status is not available. • `off` - The sensor has not been triggered. • `on` - The terminal is about to be rebooted or shutdown.
displaySensorState Specifies the state of the Consumer Display. Specified as one of the following: • `notAvailable` - The status is not available. • `off` - The Consumer Display is switched off. • `on` - The Consumer Display is in a good state and is turned on. • `displayError` - The Consumer Display is in an error state.
operatorCallButtonSensorState Specifies the state of the Operator Call Button as one of the following: • `notAvailable` - The status is not available. • `off` - The Operator Call Button is released (not pressed). • `on` - The Operator Call Button is being pressed.
handsetSensorState Specifies the state of the Handset, which is a device similar to a telephone receiver. Specified as one of the following: • `notAvailable` - The status is not available. • `onTheHook` - The Handset is on the hook. • `offTheHook` - The Handset is off the hook.
headsetMicrophoneSensorState Specifies the presence or absence of a consumer's headset microphone connected to the Microphone Jack. Specified as one of the following: • `notAvailable` - The status is not available. • `present` - There is a headset microphone connected. • `notPresent` - There is no headset microphone connected.
FasciaMicrophoneSensorState Specifies the state of the fascia microphone as one of the following: • `notAvailable` - The status is not available. • `off` - The Fascia Microphone is turned off. • `on` - The Fascia Microphone is turned on.
safeDoorState Specifies the state of the Safe Doors. Safe Doors are doors that open up for secure hardware, such as the note dispenser, the security device, etc. Specified as one of the following: • `notAvailable` - The status is not available. • `closed` - The Safe Doors are closed. • `open` - At least one of the Safe Doors is open. • `locked` - All Safe Doors are closed and locked. • `bolted` - All Safe Doors are closed, locked and bolted. • `tampered` - At least one of the Safe Doors has potentially been tampered with.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
vandalShieldState Specifies the state of the Vandal Shield. The Vandal Shield is a door that opens up for consumer access to the terminal. Specified as one of the following: • `notAvailable` - The status is not available. • `closed` - The Vandal Shield is closed. • `open` - The Vandal Shield is open. • `locked` - The Vandal Shield is closed and locked. • `service` - The Vandal Shield is in service position. • `keyboard` - The Vandal Shield position permits access to the keyboard. • `ajar` - The Vandal Shield is ajar. • `jammed` - The Vandal Shield is jammed. • `tampered` - The Vandal Shield has potentially been tampered with.
cabinetFrontDoorState Specifies the overall state of the Front Cabinet Doors. The front is defined as the side facing the customer/consumer. Cabinet Doors are doors that open up for consumables, and hardware that does not have to be in a secure place. Specified as one of the following: • `notAvailable` - The status is not available. • `closed` - All front Cabinet Doors are closed. • `open` - At least one of the front Cabinet Doors is open. • `locked` - All front Cabinet Doors are closed and locked. • `bolted` - All front Cabinet Doors are closed, locked and bolted. • `tampered` - At least one of the front Cabinet Doors has potentially been tampered with.
cabinetRearDoorState Specifies the overall state of the Rear Cabinet Doors. The front is defined as the side opposite the side facing the customer/consumer. Cabinet Doors are doors that open up for consumables, and hardware that does not have to be in a secure place. Specified as one of the following: • `notAvailable` - The status is not available. • `closed` - All rear Cabinet Doors are closed. • `open` - At least one of the rear Cabinet Doors is open. • `locked` - All rear Cabinet Doors are closed and locked. • `bolted` - All rear Cabinet Doors are closed, locked and bolted. • `tampered` - At least one of the rear Cabinet Doors has potentially been tampered with.
cabinetLeftDoorState Specifies the overall state of the Left Cabinet Doors. The left is defined as the side to the left as seen by the customer/consumer. Cabinet Doors are doors that open up for consumables, and hardware that does not have to be in a secure place. Specified as one of the following: • `notAvailable` - The status is not available. • `closed` - All left Cabinet Doors are closed. • `open` - At least one of the left Cabinet Doors is open. • `locked` - All left Cabinet Doors are closed and locked. • `bolted` - All left Cabinet Doors are closed, locked and bolted. • `tampered` - At least one of the left Cabinet Doors has potentially been tampered with.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
cabinetRightDoorState Specifies the overall state of the Right Cabinet Doors. The right is defined as the side to the right as seen by the customer/consumer. Cabinet Doors are doors that open up for consumables, and hardware that does not have to be in a secure place. Specified as one of the following: • `notAvailable` - The status is not available. • `closed` - All right Cabinet Doors are closed. • `open` - At least one of the right Cabinet Doors is open. • `locked` - All right Cabinet Doors are closed and locked. • `bolted` - All right Cabinet Doors are closed, locked and bolted. • `tampered` - At least one of the right Cabinet Doors has potentially been tampered with.
openClosedIndicatorState Specifies the state of the Open/Closed Indicator as one of the following: • `notAvailable` - The status is not available. • `closed` - The terminal is closed for a consumer. • `open` - The terminal is open to be used by a consumer.
audioState Specifies the state of the Audio Indicator as either one or a combination of the following values. Interpretation of this value is vendor-specific and therefore it is not possible to guarantee a consistent actual sound pattern across different vendor hardware: • `notAvailable` - The status is not available. • `off` - The Audio Indicator is turned off. • `keypress` - The Audio Indicator sounds a key click signal. • `exclamation` - The Audio Indicator sounds an exclamation signal. • `warning` - The Audio Indicator sounds a warning signal. • `error` - The Audio Indicator sounds an error signal. • `critical` - The Audio Indicator sounds a critical signal. • `continuous` - The Audio Indicator is turned on continuously.
audioState/notAvailable default: false
audioState/off default: false
audioState/keypress default: false
audioState/exclamation default: false
audioState/warning default: false
audioState/error default: false
audioState/critical default: false
audioState/continuous default: false

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
heatingState Specifies the state of the internal heating as one of the following: • `notAvailable` - The status is not available. • `off` - The internal heating is turned off. • `on` - The internal heating is turned on.
consumerDisplayBacklightState Specifies the Consumer Display Backlight as one of the following: • `notAvailable` - The status is not available. • `off` - The Consumer Display Backlight is turned off. • `on` - Consumer Display Backlight is turned on.
signageDisplayState Specifies the state of the Signage Display. The Signage Display is a lighted banner or marquee that can be used to display information or an advertisement. Any dynamic data displayed must be loaded by a means external to the Service Provider. Specified as one of the following: • `notAvailable` - The status is not available. • `off` - The Signage Display is turned off. • `on` - The Signage Display is turned on.
volumeState/available Specifies if the volume control is available. Possible values: • `false` - The volume control is not available. • `true` - The volume control is available. default: false

volumeState/volumeLevel
Specifies the value of the Volume Control, if available. 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. The interval is defined in logarithmic steps, e.g. a volume control on a radio. Note: The Volume Control property is vendor-specific and therefore it is not possible to guarantee a consistent actual volume level across different vendor hardware.
Property value constraints:
`minimum: 1`
`maximum: 1000`

UPSState Specifies the state of the Uninterruptible Power Supply device as a combination of the following values: • `notAvailable` - The status is not available. • `low` - The charge level of the UPS is low. • `engaged` - TThe UPS is engaged. • `powering` - The UPS is powering the system. • `recovered` - The UPS was engaged when the main power went off.
UPSState/notAvailable default: false
UPSState/low default: false
UPSState/engaged default: false

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
UPSState/powering default: false
UPSState/recovered default: false
audibleAlarmState Species the state of the Audible Alarm device as one of the following: • `notAvailable` - The status is not available. • `off` - The Alarm is turned off. • `on` - The Alarm is turned on.
enhancedAudioControlState Specifies the state of the Enhanced Audio Controller. The Enhanced Audio Controller controls how private and public audio are broadcast when the headset is inserted into/removed from the audio jack and when the handset is off-hook/on-hook. In the following, Privacy Device is used to refer to either the headset or handset. The Enhanced Audio Controller state is specified as one of the following: • `notAvailable` - The status is not available. • `publicAudioManual` - The Enhanced Audio Controller is in manual mode and is in the public state (i.e. audio will be played through speakers). Activating a Privacy Device (headset connected/handset off-hook) will have no impact, i.e. Output will remain through the speakers & no audio will be directed to the Privacy Device. • `publicAudioAuto` - The Enhanced Audio Controller is in auto mode and is in the public state (i.e. audio will be played through speakers). When a Privacy Device is activated, the device will go to the private state. • `publicAudioSemiAuto` - The Enhanced Audio Controller is in semi-auto mode and is in the public state (i.e. audio will be played through speakers). When a Privacy Device is activated, the device will go to the private state. • `privateAudioManual` - he Enhanced Audio Controller is in manual mode and is in the private state (i.e. audio will be played only through a connected Privacy Device). In private mode, no audio is transmitted through the speakers. • `privateAudioAuto` - The Enhanced Audio Controller is in auto mode and is in the private state (i.e. audio will be played only through a connected Privacy Device). In private mode, no audio is transmitted through the speakers. When a Privacy Device is deactivated (headset disconnected/handset on-hook), the device will go to the public state. Where there is more than one Privacy Device, the device will go to the public state only when all Privacy Devices have been deactivated. • `privateAudioSemiAuto` - The Enhanced Audio Controller is in semi-auto mode and is in the private state (i.e. audio will be played only through a connected Privacy Device). In private mode, no audio is transmitted through the speakers. When a Privacy Device is deactivated, the device will remain in the private state.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties

enhancedMicrophoneControlState

Specifies the state of the Enhanced Microphone Controller. The Enhanced Microphone Controller controls how private and public audio input are transmitted when the headset is inserted into/removed from the audio jack and when the handset is off-hook/on-hook. In the following, Privacy Device is used to refer to either the headset or handset. The Enhanced Microphone Controller state is specified as one of the followings:

- `notAvailable` - The status is not available.
- `publicAudioManual` - The Enhanced Microphone Controller is in manual mode and is in the public state (i.e. the microphone in the fascia is active). Activating a Privacy Device (headset connected/handset off-hook) will have no impact, i.e. input will remain through the fascia microphone and any microphone associated with the Privacy Device will not be active
- `publicAudioAuto` - The Enhanced Microphone Controller is in auto mode and is in the public state (i.e. the microphone in the fascia is active). When a Privacy Device with a microphone is activated, the device will go to the private state.
- `publicAudioSemiAuto` - The Enhanced Microphone Controller is in semi-auto mode and is in the public state (i.e. the microphone in the fascia is active). When a Privacy Device with a microphone is activated, the device will go to the private state.
- `privateAudioManual` - The Enhanced Microphone Controller is in manual mode and is in the private state (i.e. audio input will be via a microphone in the Privacy Device). In private mode, no audio input is transmitted through the fascia microphone.
- `privateAudioAuto` - The Enhanced Microphone Controller is in auto mode and is in the private state (i.e. audio input will be via a microphone in the Privacy Device). In private mode, no audio input is transmitted through the fascia microphone. When a Privacy Device with a microphone is deactivated (headset disconnected/handset on-hook), the device will go to the public state. Where there is more than one Privacy Device with a microphone, the device will go to the public state only when all such Privacy Devices have been deactivated.
- `privateAudioSemiAuto` - The Enhanced Microphone Controller is in semi-auto mode and is in the private state (i.e. audio input will be via a microphone in the Privacy Device). In private mode, no audio is transmitted through the fascia microphone. When a Privacy Device with a microphone is deactivated, the device will remain in the private state.

microphoneVolumeState/available

Specifies if the Microphone Volume Control is available. Possible values:

- `false` - The volume control is not available.
- `true` - The volume control is available.

default: false

microphoneVolumeState/volumeLevel

Specifies the value of the Microphone Volume Control, if available. 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. The interval is defined in logarithmic steps, e.g. a volume control on a radio. Note: The Microphone Volume Control property is vendor-specific and therefore it is not possible to guarantee a consistent actual volume level across different vendor hardware.

Property value constraints:

```
minimum: 1
maximum: 1000
```

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

21. Storage Interface

This chapter defines the Storage interface functionality and messages.

21.1 Summary

This specification describes the functionality of an XFS4IoT compliant Storage 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 other interfaces which require media storage functionality such as Dispenser and/or CashAcceptor interfaces to handle management of the device storage units.

21.2 General Information

21.2.1 Transaction Flows

The following sections describe how various scenarios are handled using XFS4IoT Storage.

Replenishment of a Cash Handling device

Manual Cash Replenishment in XFS4IoT is performed using the Storage.SetStorage command.

Storage.SetStorage can operate in two flows depending on whether the associated XFS4IoT service supports exchange sessions. During an exchange session, the following additional functionality applies:

- Operational commands such as dispensing notes are not allowed
- Cash configuration such as currency and value can be set. See *Storage.SetStorage* for details of which fields can be set

In flow 1, one or more cash units are replenished and as only counts have changed, an exchange session is not required. In this scenario the replenisher removes a cash unit and replaces it with one which contains 1000 USD 20 notes.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Flow 1. Simple Replenishment

In flow 2, a cash unit needs to be configured, therefore an exchange session is required. In this scenario the replenisher removes the cash unit used in flow 1 and replaces it with a different one which contains 1000 USD 20 notes.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Flow 2. Replenishment Including Exchange

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

21.3 Command Messages

21.3.1 Storage.GetStorage

This command is used to obtain information regarding the status, capabilities and contents of storage units. The capabilities of the storage unit can be used to dynamically configure the storage unit using Storage.SetStorage.

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 canceled. default: 0

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"storage": {`	object	
`"unit1": {`	object	
`"positionName": "Top Right",`	string	
`"capacity": 100,`	integer	
`"status": "ok",`	string	Yes
`"serialNumber": "ABCD1234",`	string	
`"cash": {`	object	
`"capabilities": {`	object	
`"types": {`	object	
`"cashIn": false,`	boolean	
`"cashOut": false,`	boolean	
`"replenishment": false,`	boolean	
`"cashInRetract": false,`	boolean	
`"cashOutRetract": false,`	boolean	
`"reject": false`	boolean	
`},`		

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Payload	Type	Required
"items": {	object	
"fit": false,	boolean	
"unfit": false,	boolean	
"unrecognized": false,	boolean	
"counterfeit": false,	boolean	
"suspect": false,	boolean	
"inked": false,	boolean	
"coupon": false,	boolean	
"document": false	boolean	
},		
"hardwareSensors": false,	boolean	
"retractAreas": 1,	integer	
"retractThresholds": false,	boolean	
"cashItems": {	object	
"*20USD1*": {	object	
"noteID": 1,	integer	
"currency": "USD",	string	
"value": 20,	number	
"release": 1	integer	
}		
}		
},		
"configuration": {	object	
"types": {	object	
See types properties.		
},		
"items": {	object	
See items properties.		
},		
"currency": "USD",	string	
"value": 20,	number	
"highThreshold": 500,	integer	
"lowThreshold": 10,	integer	
"appLockIn": false,	boolean	
"appLockOut": false,	boolean	
"cashItems": {	object	
See cashItems properties.		

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Payload	Type	Required
`},`		
`"name": "$10",`	string	
`"maxRetracts": 5`	integer	
`},`		
`"status": {`	object	
`"index": 1,`	integer	
`"initial": {`	object	
`"unrecognized": 0,`	integer	
`"cash": {`	object	
`"20USD1": {`	object	
`"fit": 0,`	integer	
`"unfit": 0,`	integer	
`"suspect": 0,`	integer	
`"counterfeit": 0,`	integer	
`"inked": 0`	integer	
`}`		
`}`		
`},`		
`"out": {`	object	
`"presented": {`	object	
`See initial properties.`		
`},`		
`"rejected": {`	object	
`See initial properties.`		
`},`		
`"distributed": {`	object	
`See initial properties.`		
`},`		
`"unknown": {`	object	
`See initial properties.`		
`},`		
`"stacked": {`	object	
`See initial properties.`		
`},`		
`"diverted": {`	object	
`See initial properties.`		
`},`		

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Payload	Type	Required
"transport": {	object	
See initial properties.		
}		
},		
"in": {	object	
"retractOperations": 0,	integer	
"deposited": {	object	
See initial properties.		
},		
"retracted": {	object	
See initial properties.		
},		
"rejected": {	object	
See initial properties.		
},		
"distributed": {	object	
See initial properties.		
}		
},		
"count": 0,	integer	
"accuracy": "notSupported",	string	
"replenishmentStatus": "ok"	string	
}		
},		
"card": {	object	
"capabilities": {	object	
"type": "retain",	string	
"hardwareSensors": false	boolean	
},		
"configuration": {	object	
"cardID": "LoyaltyCard",	string	
"threshold": 0	integer	
},		
"status": {	object	
"initialCount": 0,	integer	
"count": 0,	integer	
"retainCount": 0,	integer	

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Payload	Type	Required
`"replenishmentStatus": "ok"`	string	
`}`		
`}`		
`},`		
`"unit2": {`	object	
`See unit1 properties.`		
`}`		
`}`		
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

storage
Object containing storage unit information. The property name is the storage unit identifier.

storage/unit1 (example name)
The object contains a single storage unit.
Property name constraints:
```
pattern: ^.{1,5}$
```

storage/unit1/positionName
Fixed physical name for the position.

storage/unit1/capacity
The nominal capacity of the unit. This may be an estimate as the quality and thickness of the items stored in the unit may affect how many items can be stored. 0 means the capacity is unknown.
Property value constraints:
```
minimum: 0
```

storage/unit1/status
The state of the unit. The following values are possible:

- `ok` - The storage unit is in a good state.
- `inoperative` - The storage unit is inoperative.
- `missing` - The storage unit is missing.
- `notConfigured` - The storage unit has not been configured for use.
- `manipulated` - The storage unit has been inserted (including removal followed by a reinsertion) when the device was not in the exchange state - see Storage.StartExchange. This storage unit cannot be used. Only applies to services which support the exchange state.

storage/unit1/serialNumber
The storage unit's serial number if it can be read electronically.

storage/unit1/cash
The cash related contents, status and configuration of the unit.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage/unit1/cash/capabilities Indicates what the storage unit is capable of - this includes information which is a combination of that reported in WFS_INF_CDM_CASH_UNIT_INFO, WFS_INF_CIM_CASH_UNIT_INFO and WFS_INF_CIM_CASH_UNIT_CAPABILITIES in XFS 3.x.
storage/unit1/cash/capabilities/types The types of operation the unit is capable of or configured to perform. This is a combination of one or more operations. May only be modified in an exchange state if applicable.
storage/unit1/cash/capabilities/types/cashIn The unit can accept cash items. If *cashOut* is also true then the unit can recycle.
storage/unit1/cash/capabilities/types/cashOut The unit can dispense cash items. If *cashIn* is also true then the unit can recycle.
storage/unit1/cash/capabilities/types/replenishment Replenishment container. A storage unit can be refilled from or emptied to a replenishment container.
storage/unit1/cash/capabilities/types/cashInRetract Retract unit. Items can be retracted into this unit during Cash In operations.
storage/unit1/cash/capabilities/types/cashOutRetract Retract unit. Items can be retracted into this unit during Cash Out operations.
storage/unit1/cash/capabilities/types/reject Reject unit. Items can be rejected into this unit.
storage/unit1/cash/capabilities/items The types of cash media the unit is capable of storing or configured to store. This is a combination of one or more item types. May only be modified in an exchange state if applicable. See Note Classification for more information about cash classification levels.
storage/unit1/cash/capabilities/items/fit The storage unit can store cash items which are fit for recycling (classification level 4).
storage/unit1/cash/capabilities/items/unfit The storage unit can store cash items which are unfit for recycling (classification level 4).
storage/unit1/cash/capabilities/items/unrecognized The storage unit can store unrecognized cash items (classification level 1).
storage/unit1/cash/capabilities/items/counterfeit The storage unit can store counterfeit cash items (classification level 2).
storage/unit1/cash/capabilities/items/suspect The storage unit can store suspect cash items (classification level 3).
storage/unit1/cash/capabilities/items/inked The storage unit can store cash items which have been identified as ink stained.
storage/unit1/cash/capabilities/items/coupon Storage unit containing coupons or advertising material.
storage/unit1/cash/capabilities/items/document Storage unit containing documents.
storage/unit1/cash/capabilities/hardwareSensors Indicates whether the storage unit has sensors which report the status.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage/unit1/cash/capabilities/retractAreas If items can be retracted into this storage unit, this is the number of areas within the storage unit which allow physical separation of different bunches. If there is no physical separation of retracted bunches within this storage unit, this value is 1. Property value constraints: `minimum: 1`
storage/unit1/cash/capabilities/retractThresholds If true, indicates that retract capacity is based on counts. If false, indicates that retract capacity is based on the number of commands which resulted in items being retracted into the storage unit.
storage/unit1/cash/capabilities/cashItems Lists the cash items which the storage unit is physically capable of handling. For example a coin storage unit may be restricted to one coin denomination due to the hardware.
storage/unit1/cash/capabilities/cashItems/20USD1 (example name) An object containing information about a single cash item supported by the device.
storage/unit1/cash/capabilities/cashItems/20USD1/noteID Assigned by the XFS4IoT service. A unique number identifying a single cash item. Each unique combination of the other properties will have a different noteID. Can be used for migration of *usNoteID* from XFS 3.x. Property value constraints: `minimum: 1`
storage/unit1/cash/capabilities/cashItems/20USD1/currency ISO 4217 currency. Property value constraints: `maxLength: 3`
storage/unit1/cash/capabilities/cashItems/20USD1/value Absolute value of all contents, 0 if mixed. May only be modified in an exchange state if applicable. May be a floating point value to allow for coins and notes which have a value which is not a whole multiple of the currency unit.
storage/unit1/cash/capabilities/cashItems/20USD1/release The release of the cash item. The higher this number is, the newer the release. If zero or not reported, there is only one release of that cash item or the device is not capable of distinguishing different release of the item, for example in a simple cash dispenser. An example of how this can be used is being able to sort different releases of the same denomination note to different storage units to take older notes out of circulation. This value is device, banknote reader and currency dependent, therefore a release number of the same cash item will not necessarily have the same value in different systems and any such usage would be specific to a specific device's configuration. Property value constraints: `minimum: 0`
storage/unit1/cash/configuration Indicates what this storage unit is configured as or is being configured to do - where applicable the supported options can be derived from capabilities. If the service supports an exchange state, only a subset of these parameters may be modified unless in an exchange. Parameters which may only be modified in an exchange state are listed.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage/unit1/cash/configuration/types The types of operation the unit is capable of or configured to perform. This is a combination of one or more operations. May only be modified in an exchange state if applicable.
storage/unit1/cash/configuration/items The types of cash media the unit is capable of storing or configured to store. This is a combination of one or more item types. May only be modified in an exchange state if applicable. See Note Classification for more information about cash classification levels.
storage/unit1/cash/configuration/currency ISO 4217 currency. May only be modified in an exchange state if applicable. Property value constraints: `maxLength: 3`
storage/unit1/cash/configuration/highThreshold If specified, replenishmentStatus is set to *high* if count is greater than this number. If not specified, high is based on hardware sensors if supported - see hardwareSensors. Property value constraints: `minimum: 1`
storage/unit1/cash/configuration/lowThreshold If specified, replenishmentStatus is set to *low* if count is less than this number. If not specified, low is based on hardware sensors if supported - see hardwareSensors. Property value constraints: `minimum: 1`
storage/unit1/cash/configuration/appLockIn If true, items cannot be accepted into the storage unit in Cash In operations.
storage/unit1/cash/configuration/appLockOut If true, items cannot be dispensed from the storage unit in Cash Out operations.
storage/unit1/cash/configuration/cashItems An object containing multiple cash items, listing what a storage unit is capable of or configured to handle.
storage/unit1/cash/configuration/name Application configured name of the unit.
storage/unit1/cash/configuration/maxRetracts If specified, this is the number of retract operations allowed into the unit. If not specified, the maximum number is not limited by counts. Property value constraints: `minimum: 1`
storage/unit1/cash/status Indicates the storage unit status - this includes information which is a combination of that reported in WFS_INF_CDM_CASH_UNIT_INFO and WFS_INF_CIM_CASH_UNIT_INFO in XFS 3.x.
storage/unit1/cash/status/index Assigned by the XFS4IoT service. Will be a unique number which can be used to determine *usNumber* in XFS 3.x migration. This can change as cash storage units are added and removed from the storage collection. Property value constraints: `minimum: 1`

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage/unit1/cash/status/initial The cash related items which were in the storage unit at the last replenishment.
storage/unit1/cash/status/initial/unrecognized Count of unrecognized items handled by the cash interface
storage/unit1/cash/status/initial/cash Counts of cash items broken down by cash item type and classification
storage/unit1/cash/status/initial/cash/20USD1 (example name) Counts of a given CashItem broken down by classification.
storage/unit1/cash/status/initial/cash/20USD1/fit Count of genuine cash items which are fit for recycling (classification level 4 - see Note Classification).
storage/unit1/cash/status/initial/cash/20USD1/unfit Count of genuine cash items which are unfit for recycling (classification level 4 - see Note Classification).
storage/unit1/cash/status/initial/cash/20USD1/suspect Count of suspect cash items (classification level 3 - see Note Classification).
storage/unit1/cash/status/initial/cash/20USD1/counterfeit Count of counterfeit cash items (classification level 2 - see Note Classification).
storage/unit1/cash/status/initial/cash/20USD1/inked Count of cash items which have been identified as ink stained.
storage/unit1/cash/status/out The items moved from this storage unit by cash commands to another destination since the last replenishment of this unit. Reset to empty if initial is set for this unit by Storage.GetStorage.
storage/unit1/cash/status/out/presented The items dispensed from this storage unit which are or were customer accessible.
storage/unit1/cash/status/out/rejected The items dispensed from this storage unit which were invalid and were diverted to a reject storage unit and were not customer accessible during the operation.
storage/unit1/cash/status/out/distributed The items dispensed from this storage unit which were moved to a storage unit other than a reject storage unit and were not customer accessible during the operation.
storage/unit1/cash/status/out/unknown The items dispensed from this storage unit which moved to an unknown position.
storage/unit1/cash/status/out/stacked The items dispensed from this storage unit which are not customer accessible and are currently stacked awaiting presentation to the customer. This item list can increase and decrease as items are moved around in the device.
storage/unit1/cash/status/out/diverted The items dispensed from this storage unit which are not customer accessible and were diverted to a temporary location due to being invalid and have not yet been deposited in a storage unit. This item list can increase and decrease as items are moved around in the device.
storage/unit1/cash/status/out/transport The items dispensed from this storage unit which are not customer accessible and which have jammed in the transport. This item list can increase and decrease as items are moved around in the device.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage/unit1/cash/status/in List of items inserted in this storage unit by cash commands from another source since the last replenishment of this unit. Reset to empty if initial is set for this unit by Storage.GetStorage.
storage/unit1/cash/status/in/retractOperations Number of cash retract operations which resulted in items entering this storage unit. This can be used where devices do not have the capability to count or validate items after presentation.
storage/unit1/cash/status/in/deposited The items deposited in the storage unit during a Cash In transaction.
storage/unit1/cash/status/in/retracted The items deposited in the storage unit after being accessible to a customer. This may be inaccurate or not counted if items are not counted or re-validated after presentation, the number of retract operations is also reported separately in *retractOperations*.
storage/unit1/cash/status/in/rejected The items deposited in this storage unit originating from another storage unit but rejected due to being invalid. This count may be inaccurate due to the nature of rejected items.
storage/unit1/cash/status/in/distributed The items deposited in this storage unit originating from another storage unit but not rejected.
storage/unit1/cash/status/count Total count of the items in the unit, derived from the initial, out and in counts, but supplied for ease of use. For units which dispense items, this count is only decremented when the items are either known to be in customer access or successfully rejected, therefore the intermediate *out* fields are not included in this calculation: • stacked • transport • unknown If counts being incorrectly set at replenishment time means that this would result in a negative number, this reports 0. Property value constraints: `minimum: 0`
storage/unit1/cash/status/accuracy Describes the accuracy of *count*. Following values are possible: • `notSupported` - The hardware is not capable of determining the accuracy of *count*. • `accurate` - The *count* is expected to be accurate. The notes were previously counted and there have since been no events that might have introduced inaccuracy. • `accurateSet` - The *count* is expected to be accurate. The counts 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 storage unit insertion or some other hardware event.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage/unit1/cash/status/replenishmentStatus The state of the media in the unit if it can be determined. Note that overall status of the storage unit must be taken into account when deciding whether the storage unit is usable and whether replenishment status is applicable. In particular, if the overall status is *missing* this will not be reported. The following values are possible: • `ok` - The storage unit media is in a good state. • `full` - The storage unit is full. • `high` - The storage unit is almost full (either sensor based or exceeded the highThreshold. • `low` - The storage unit is almost empty (either sensor based or below the lowThreshold). • `empty` - The storage unit is empty, or insufficient items in the storage unit are preventing further dispense operations.
storage/unit1/card The card related contents, status and configuration of the unit.
storage/unit1/card/capabilities Indicates the card storage unit capabilities.
storage/unit1/card/capabilities/type The type of card storage as one of the following: • `retain` - The storage unit can retain cards. • `dispense` - The storage unit can dispense cards. • `park` - The storage unit can be used to temporarily store a card allowing another card to enter the transport.
storage/unit1/card/capabilities/hardwareSensors The storage unit has hardware sensors that can detect threshold states. default: false
storage/unit1/card/configuration Indicates the card storage unit configuration.
storage/unit1/card/configuration/cardID The identifier that may be used to identify the type of cards in the storage unit. This is only applicable to *dispense* type storage units.
storage/unit1/card/configuration/threshold If the threshold value is non zero, hardware sensors in the storage unit do not trigger Storage.StorageThresholdEvent events. If non zero, when *count* reaches the threshold value: • For retain type storage units, a high threshold will be sent. • For dispense type storage units, a low threshold will be sent. Property value constraints: `minimum: 0`
storage/unit1/card/status Indicates the card storage unit status.
storage/unit1/card/status/initialCount The initial number of cards in the storage unit. This is only applicable to dispense type storage units. This value is persistent. Property value constraints: `minimum: 0`

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage/unit1/card/status/count The number of cards in the storage unit. If the storage unit type is dispense: • This count also includes a card dispensed from the storage unit which has not been moved to either the exit position or a dispense type storage unit. • This count is decremented when a card from the card storage unit is moved to the exit position or retained. If this value reaches zero it will not decrement further but will remain at zero. If the storage unit type is retain: • The count is incremented when a card is moved into the storage unit. If the storage unit type is park: • The count will increment when a card is moved into the storage module and decremented when a card is moved out of the storage module. This value is persistent. Property value constraints: `minimum: 0`
storage/unit1/card/status/retainCount The number of cards from this storage unit which are in a retain storage unit. This is only applicable to dispense type storage units. This value is persistent. Property value constraints: `minimum: 0`
storage/unit1/card/status/replenishmentStatus The state of the cards in the storage unit if it can be determined. Note that overall status of the storage unit must be taken into account when deciding whether the storage unit is usable and whether replenishment status is applicable. In particular, if the overall status is *missing* this will be omitted. The following values are possible: • `ok` - The storage unit is in a good state. • `full` - The storage unit is full. • `high` - The storage unit is almost full (either sensor based or above the threshold). • `low` - The storage unit is almost empty (either sensor based or below the threshold). • `empty` - The storage unit is empty.

Event Messages

None

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

21.3.2 Storage.SetStorage

This command is used to adjust information about the configuration and contents of the device's storage units. 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 omitted.

This command generates the Storage.StorageChangedEvent to inform applications that storage unit information has been changed.

Only a subset of the information reported by Storage.GetStorage may be modified by this command therefore the payload is a subset of the GetStorage output. In addition, if the service supports an exchange state, only a subset of the information which may be modified by this command can be modified unless the service is in an exchange state. The descriptions of each field list which can be modified at any point using this command; 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	
`"storage": {`	object	
`"unit1": {`	object	
`"cash": {`	object	
`"configuration": {`	object	
`"types": {`	object	
`"cashIn": false,`	boolean	
`"cashOut": false,`	boolean	
`"replenishment": false,`	boolean	
`"cashInRetract": false,`	boolean	
`"cashOutRetract": false,`	boolean	
`"reject": false`	boolean	
`},`		
`"items": {`	object	
`"fit": false,`	boolean	
`"unfit": false,`	boolean	
`"unrecognized": false,`	boolean	
`"counterfeit": false,`	boolean	
`"suspect": false,`	boolean	
`"inked": false,`	boolean	
`"coupon": false,`	boolean	
`"document": false`	boolean	
`},`		
`"currency": "USD",`	string	

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Payload	Type	Required
"value": 20,	number	
"highThreshold": 500,	integer	
"lowThreshold": 10,	integer	
"appLockIn": false,	boolean	
"appLockOut": false,	boolean	
"cashItems": {	object	
"20USD1": {	object	
"noteID": 1,	integer	
"currency": "USD",	string	
"value": 20,	number	
"release": 1	integer	
}		
},		
"name": "$10",	string	
"maxRetracts": 5	integer	
},		
status": {	object	
"initial": {	object	
"unrecognized": 0,	integer	
"cash": {	object	
"20USD1": {	object	
"fit": 0,	integer	
"unfit": 0,	integer	
"suspect": 0,	integer	
"counterfeit": 0,	integer	
"inked": 0	integer	
}		
}		
}		
}		
},		
"card": {	object	
"configuration": {	object	
"cardID": "LoyaltyCard",	string	
"threshold": 0	integer	
},		
"status": {	object	

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Payload	Type	Required
`"initialCount": 0`	integer	
`}`		
`}`		
`},`		
`"unit2": {`	object	
`See unit1 properties.`		
`}`		
`}`		
`}`		

Properties
timeout Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be canceled. default: 0
storage Object containing storage unit information.
storage/unit1 (example name) The object contains a single storage unit. Property name constraints: `pattern: ^.{1,5}$`
storage/unit1/cash The cash related contents and configuration of the unit.
storage/unit1/cash/configuration Indicates what this storage unit is configured as or is being configured to do - where applicable the supported options can be derived from capabilities. If the service supports an exchange state, only a subset of these parameters may be modified unless in an exchange. Parameters which may only be modified in an exchange state are listed.
storage/unit1/cash/configuration/types The types of operation the unit is capable of or configured to perform. This is a combination of one or more operations. May only be modified in an exchange state if applicable.
storage/unit1/cash/configuration/types/cashIn The unit can accept cash items. If *cashOut* is also true then the unit can recycle.
storage/unit1/cash/configuration/types/cashOut The unit can dispense cash items. If *cashIn* is also true then the unit can recycle.
storage/unit1/cash/configuration/types/replenishment Replenishment container. A storage unit can be refilled from or emptied to a replenishment container.
storage/unit1/cash/configuration/types/cashInRetract Retract unit. Items can be retracted into this unit during Cash In operations.
storage/unit1/cash/configuration/types/cashOutRetract Retract unit. Items can be retracted into this unit during Cash Out operations.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage/unit1/cash/configuration/types/reject Reject unit. Items can be rejected into this unit.
storage/unit1/cash/configuration/items The types of cash media the unit is capable of storing or configured to store. This is a combination of one or more item types. May only be modified in an exchange state if applicable. See Note Classification for more information about cash classification levels.
storage/unit1/cash/configuration/items/fit The storage unit can store cash items which are fit for recycling (classification level 4).
storage/unit1/cash/configuration/items/unfit The storage unit can store cash items which are unfit for recycling (classification level 4).
storage/unit1/cash/configuration/items/unrecognized The storage unit can store unrecognized cash items (classification level 1).
storage/unit1/cash/configuration/items/counterfeit The storage unit can store counterfeit cash items (classification level 2).
storage/unit1/cash/configuration/items/suspect The storage unit can store suspect cash items (classification level 3).
storage/unit1/cash/configuration/items/inked The storage unit can store cash items which have been identified as ink stained.
storage/unit1/cash/configuration/items/coupon Storage unit containing coupons or advertising material.
storage/unit1/cash/configuration/items/document Storage unit containing documents.
storage/unit1/cash/configuration/currency ISO 4217 currency. May only be modified in an exchange state if applicable. Property value constraints: `maxLength: 3`
storage/unit1/cash/configuration/value Absolute value of all contents, 0 if mixed. May only be modified in an exchange state if applicable. May be a floating point value to allow for coins and notes which have a value which is not a whole multiple of the currency unit.
storage/unit1/cash/configuration/highThreshold If specified, replenishmentStatus is set to *high* if count is greater than this number. If not specified, high is based on hardware sensors if supported - see hardwareSensors. Property value constraints: `minimum: 1`
storage/unit1/cash/configuration/lowThreshold If specified, replenishmentStatus is set to *low* if count is less than this number. If not specified, low is based on hardware sensors if supported - see hardwareSensors. Property value constraints: `minimum: 1`
storage/unit1/cash/configuration/appLockIn If true, items cannot be accepted into the storage unit in Cash In operations.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage/unit1/cash/configuration/appLockOut If true, items cannot be dispensed from the storage unit in Cash Out operations.
storage/unit1/cash/configuration/cashItems An object containing multiple cash items, listing what a storage unit is capable of or configured to handle.
storage/unit1/cash/configuration/cashItems/20USD1 (example name) An object containing information about a single cash item supported by the device.
storage/unit1/cash/configuration/cashItems/20USD1/noteID Assigned by the XFS4IoT service. A unique number identifying a single cash item. Each unique combination of the other properties will have a different noteID. Can be used for migration of *usNoteID* from XFS 3.x. Property value constraints: `minimum: 1`
storage/unit1/cash/configuration/cashItems/20USD1/currency ISO 4217 currency. Property value constraints: `maxLength: 3`
storage/unit1/cash/configuration/cashItems/20USD1/release The release of the cash item. The higher this number is, the newer the release. If zero or not reported, there is only one release of that cash item or the device is not capable of distinguishing different release of the item, for example in a simple cash dispenser. An example of how this can be used is being able to sort different releases of the same denomination note to different storage units to take older notes out of circulation. This value is device, banknote reader and currency dependent, therefore a release number of the same cash item will not necessarily have the same value in different systems and any such usage would be specific to a specific device's configuration. Property value constraints: `minimum: 0`
storage/unit1/cash/configuration/name Application configured name of the unit.
storage/unit1/cash/configuration/maxRetracts If specified, this is the number of retract operations allowed into the unit. If not specified, the maximum number is not limited by counts. Property value constraints: `minimum: 1`
storage/unit1/cash/status/initial The cash related items which are in the storage unit at the last replenishment. If specified, out and in are reset to empty.
storage/unit1/cash/status/initial/unrecognized Count of unrecognized items handled by the cash interface
storage/unit1/cash/status/initial/cash Counts of cash items broken down by cash item type and classification
storage/unit1/cash/status/initial/cash/20USD1 (example name) Counts of a given CashItem broken down by classification.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage/unit1/cash/status/initial/cash/20USD1/fit Count of genuine cash items which are fit for recycling (classification level 4 - see Note Classification).
storage/unit1/cash/status/initial/cash/20USD1/unfit Count of genuine cash items which are unfit for recycling (classification level 4 - see Note Classification).
storage/unit1/cash/status/initial/cash/20USD1/suspect Count of suspect cash items (classification level 3 - see Note Classification).
storage/unit1/cash/status/initial/cash/20USD1/counterfeit Count of counterfeit cash items (classification level 2 - see Note Classification).
storage/unit1/cash/status/initial/cash/20USD1/inked Count of cash items which have been identified as ink stained.
storage/unit1/card The card related contents and configuration of the unit.
storage/unit1/card/configuration Indicates the card storage unit configuration.
storage/unit1/card/configuration/cardID The identifier that may be used to identify the type of cards in the storage unit. This is only applicable to *dispense* type storage units.
storage/unit1/card/configuration/threshold If the threshold value is non zero, hardware sensors in the storage unit do not trigger Storage.StorageThresholdEvent events. If non zero, when *count* reaches the threshold value: • For retain type storage units, a high threshold will be sent. • For dispense type storage units, a low threshold will be sent. Property value constraints: `minimum: 0`
storage/unit1/card/status Indicates the card storage unit status being set.
storage/unit1/card/status/initialCount The number of cards in the storage unit at the last replenishment. If specified, count is set to match this value and retainCount is set to zero. Property value constraints: `minimum: 0`

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "invalidUnit"`	string	
`}`		

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
completionCode The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
errorCode Specifies the error code if applicable. Following values are possible: • `invalidUnit` - Invalid unit. • `noExchangeActive` - The device is not in an exchange state and a request has been made to modify information which can only be modified in an exchange state. • `storageUnitError` - A problem occurred with a storage unit. A Storage.StorageErrorEvent will be posted with the details.

Event Messages

- Storage.StorageChangedEvent
- Storage.StorageErrorEvent
- Storage.StorageThresholdEvent

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

21.3.3 Storage.StartExchange

This command puts the device in an **exchange** state, i.e. a state in which storage units can be emptied, replenished, removed or replaced. The command will initiate any physical processes which may be necessary to make the storage units accessible. If this command returns a successful completion the device is in an exchange state.

The current exchange state is reported by exchange and any change of state is marked by a Common.ExchangeStateChangedEvent.

While in the exchange state:

- Storage.SetStorage may be called as required to configure the storage units. Note that some of the storage fields may only be set while in an exchange state, particularly fields which modify the configuration of the storage unit or units. The fields affected by this are documented in *Storage.SetStorage*. Note that *Storage.SetStorage* does not need to be called if the Service can obtain storage unit information from self-configuring units.
- Commands which operate the device mechanically such as an attempt to dispense notes may be rejected with *exchangeActive*. This allows the device to be replenished safely and in a controlled manner.

Not all devices which support the Storage interface support an exchange state, *Storage.SetStorage* may be sufficient to configure those storage units. In such devices, this command is not supported. Similarly, devices which support the Storage interface may not require an exchange state to be entered if for example only modifying counts.

The exchange state is exited by calling Storage.EndExchange.

In the exchange state the *Storage.SetStorage* command can be used multiple times to adjust the cash unit information, until the *Storage.EndExchange* command is performed.

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 canceled.
default: 0

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "storageUnitError"`	string	
`}`		

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
completionCode The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
errorCode Specifies the error code if applicable. Following values are possible: • `storageUnitError` - An error occurred with a storage unit while performing the exchange operation. A Storage.StorageErrorEvent will be sent with the details. • `exchangeActive` - The device is already in an exchange state. • `transactionActive` - A transaction is active.

Event Messages

- Storage.StorageErrorEvent
- Common.ExchangeStateChangedEvent

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

21.3.4 Storage.EndExchange

This command will end the exchange state. If any physical action took place as a result of the Storage.StartExchange command then this command will cause the cash units to be returned to their normal physical state. Any necessary device testing will also be initiated.

The current exchange state is reported by exchange and any change of state is marked by a Common.ExchangeStateChangedEvent.

Storage.SetStorage does not need to be called if the Service can obtain storage unit information from self-configuring units.

If an error occurs during the execution of this command, then the application must issue a Storage.GetStorage to determine the storage unit information.

A Storage.StorageUnitErrorEvent will be sent for any storage unit which cannot be successfully updated. If no units could be updated then a error code will be returned.

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 canceled.
default: 0

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"errorCode": "storageUnitError"`	string	
`}`		

Properties

completionCode
The completion code. If the value is *commandErrorCode*, the *errorCode* property contains the command specific completion error code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
errorCode Specifies the error code if applicable. Following values are possible: • `storageUnitError` - A storage unit problem occurred that meant no storage units could be updated. One or more Storage.StorageErrorEvent events will be sent with the details. • `noExchangeActive` - There is no exchange active.

Event Messages

- Storage.StorageChangedEvent
- Storage.StorageErrorEvent
- Storage.StorageThresholdEvent
- Common.ExchangeStateChangedEvent

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

21.4 Event Messages

21.4.1 Common.ExchangeStateChangedEvent

This event reports that the service's exchange state has changed.

Payload	Type	Required
`{`		
`"exchange": "notSupported"`	string	
`}`		

Properties

exchange

Specifies the exchange state of the service. Exchange can used to perform a manual replenishment of a device and is entered by Storage.StartExchange and completed by Storage.EndExchange. When the state changes a Common.ExchangeStateChangedEvent is posted. Following values are possible:

- `notSupported` - Exchange is not supported on this service.
- `active` - Exchange is active on this service. Commands which interact with the device may be rejected with an error code as appropriate.
- `inactive` - Exchange is not active on this service.

default: "notSupported"

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

21.5 Unsolicited Messages

21.5.1 Storage.StorageChangedEvent

This event is generated when a storage unit changes under the following circumstances:

- When the unit changes in any way due to a Storage.SetStorage command.
- When any change is made other than to counts by any other command or external intervention.

If a new storage unit is inserted the storage unit structure reported by the last Storage.GetStorage command is no longer valid. In that case an application should issue a Storage.GetStorage command after receiving this event to obtain updated storage unit information.

Payload	Type	Required
{		
"storage": {	object	
"unit1": {	object	
"positionName": "Top Right",	string	
"capacity": 100,	integer	
"status": "ok",	string	Yes
"serialNumber": "ABCD1234",	string	
"cash": {	object	
"capabilities": {	object	
"types": {	object	
"cashIn": false,	boolean	
"cashOut": false,	boolean	
"replenishment": false,	boolean	
"cashInRetract": false,	boolean	
"cashOutRetract": false,	boolean	
"reject": false	boolean	
},		
"items": {	object	
"fit": false,	boolean	
"unfit": false,	boolean	
"unrecognized": false,	boolean	
"counterfeit": false,	boolean	
"suspect": false,	boolean	
"inked": false,	boolean	
"coupon": false,	boolean	
"document": false	boolean	
},		
"hardwareSensors": false,	boolean	

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Payload	Type	Required
"retractAreas": 1,	integer	
"retractThresholds": false,	boolean	
"cashItems": {	object	
"20USD1": {	object	
"noteID": 1,	integer	
"currency": "USD",	string	
"value": 20,	number	
"release": 1	integer	
}		
}		
},		
"configuration": {	object	
"types": {	object	
See types properties.		
},		
"items": {	object	
See items properties.		
},		
"currency": "USD",	string	
"value": 20,	number	
"highThreshold": 500,	integer	
"lowThreshold": 10,	integer	
"appLockIn": false,	boolean	
"appLockOut": false,	boolean	
"cashItems": {	object	
See cashItems properties.		
},		
"name": "$10",	string	
"maxRetracts": 5	integer	
},		
"status": {	object	
"index": 1,	integer	
"initial": {	object	
"unrecognized": 0,	integer	
"cash": {	object	
"20USD1": {	object	
"fit": 0,	integer	

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Payload	Type	Required
"unfit": 0,	integer	
"suspect": 0,	integer	
"counterfeit": 0,	integer	
"inked": 0	integer	
}		
}		
},		
"out": {	object	
"presented": {	object	
See initial properties.		
},		
"rejected": {	object	
See initial properties.		
},		
"distributed": {	object	
See initial properties.		
},		
"unknown": {	object	
See initial properties.		
},		
"stacked": {	object	
See initial properties.		
},		
"diverted": {	object	
See initial properties.		
},		
"transport": {	object	
See initial properties.		
}		
},		
"in": {	object	
"retractOperations": 0,	integer	
"deposited": {	object	
See initial properties.		
},		
"retracted": {	object	
See initial properties.		

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Payload	Type	Required
`},`		
`"rejected": {`	object	
`See initial properties.`		
`},`		
`"distributed": {`	object	
`See initial properties.`		
`}`		
`},`		
`"count": 0,`	integer	
`"accuracy": "notSupported",`	string	
`"replenishmentStatus": "ok"`	string	
`}`		
`},`		
`"card": {`	object	
`"capabilities": {`	object	
`"type": "retain",`	string	
`"hardwareSensors": false`	boolean	
`},`		
`"configuration": {`	object	
`"cardID": "LoyaltyCard",`	string	
`"threshold": 0`	integer	
`},`		
`"status": {`	object	
`"initialCount": 0,`	integer	
`"count": 0,`	integer	
`"retainCount": 0,`	integer	
`"replenishmentStatus": "ok"`	string	
`}`		
`}`		
`},`		
`"unit2": {`	object	
`See unit1 properties.`		
`}`		
`}`		
`}`		

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage Object containing storage unit information. The property name is the storage unit identifier.
storage/unit1 (example name) The object contains a single storage unit. Property name constraints: `pattern: ^.{1,5}$`
storage/unit1/positionName Fixed physical name for the position.
storage/unit1/capacity The nominal capacity of the unit. This may be an estimate as the quality and thickness of the items stored in the unit may affect how many items can be stored. 0 means the capacity is unknown. Property value constraints: `minimum: 0`
storage/unit1/status The state of the unit. The following values are possible: • `ok` - The storage unit is in a good state. • `inoperative` - The storage unit is inoperative. • `missing` - The storage unit is missing. • `notConfigured` - The storage unit has not been configured for use. • `manipulated` - The storage unit has been inserted (including removal followed by a reinsertion) when the device was not in the exchange state - see Storage.StartExchange. This storage unit cannot be used. Only applies to services which support the exchange state.
storage/unit1/serialNumber The storage unit's serial number if it can be read electronically.
storage/unit1/cash The cash related contents, status and configuration of the unit.
storage/unit1/cash/capabilities Indicates what the storage unit is capable of - this includes information which is a combination of that reported in WFS_INF_CDM_CASH_UNIT_INFO, WFS_INF_CIM_CASH_UNIT_INFO and WFS_INF_CIM_CASH_UNIT_CAPABILITIES in XFS 3.x.
storage/unit1/cash/capabilities/types The types of operation the unit is capable of or configured to perform. This is a combination of one or more operations. May only be modified in an exchange state if applicable.
storage/unit1/cash/capabilities/types/cashIn The unit can accept cash items. If *cashOut* is also true then the unit can recycle.
storage/unit1/cash/capabilities/types/cashOut The unit can dispense cash items. If *cashIn* is also true then the unit can recycle.
storage/unit1/cash/capabilities/types/replenishment Replenishment container. A storage unit can be refilled from or emptied to a replenishment container.
storage/unit1/cash/capabilities/types/cashInRetract Retract unit. Items can be retracted into this unit during Cash In operations.
storage/unit1/cash/capabilities/types/cashOutRetract Retract unit. Items can be retracted into this unit during Cash Out operations.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage/unit1/cash/capabilities/types/reject Reject unit. Items can be rejected into this unit.
storage/unit1/cash/capabilities/items The types of cash media the unit is capable of storing or configured to store. This is a combination of one or more item types. May only be modified in an exchange state if applicable. See Note Classification for more information about cash classification levels.
storage/unit1/cash/capabilities/items/fit The storage unit can store cash items which are fit for recycling (classification level 4).
storage/unit1/cash/capabilities/items/unfit The storage unit can store cash items which are unfit for recycling (classification level 4).
storage/unit1/cash/capabilities/items/unrecognized The storage unit can store unrecognized cash items (classification level 1).
storage/unit1/cash/capabilities/items/counterfeit The storage unit can store counterfeit cash items (classification level 2).
storage/unit1/cash/capabilities/items/suspect The storage unit can store suspect cash items (classification level 3).
storage/unit1/cash/capabilities/items/inked The storage unit can store cash items which have been identified as ink stained.
storage/unit1/cash/capabilities/items/coupon Storage unit containing coupons or advertising material.
storage/unit1/cash/capabilities/items/document Storage unit containing documents.
storage/unit1/cash/capabilities/hardwareSensors Indicates whether the storage unit has sensors which report the status.
storage/unit1/cash/capabilities/retractAreas If items can be retracted into this storage unit, this is the number of areas within the storage unit which allow physical separation of different bunches. If there is no physical separation of retracted bunches within this storage unit, this value is 1. Property value constraints: `minimum: 1`
storage/unit1/cash/capabilities/retractThresholds If true, indicates that retract capacity is based on counts. If false, indicates that retract capacity is based on the number of commands which resulted in items being retracted into the storage unit.
storage/unit1/cash/capabilities/cashItems Lists the cash items which the storage unit is physically capable of handling. For example a coin storage unit may be restricted to one coin denomination due to the hardware.
storage/unit1/cash/capabilities/cashItems/20USD1 (example name) An object containing information about a single cash item supported by the device.
storage/unit1/cash/capabilities/cashItems/20USD1/noteID Assigned by the XFS4IoT service. A unique number identifying a single cash item. Each unique combination of the other properties will have a different noteID. Can be used for migration of *usNoteID* from XFS 3.x. Property value constraints: `minimum: 1`

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage/unit1/cash/capabilities/cashItems/20USD1/currency ISO 4217 currency. Property value constraints: `maxLength: 3`
storage/unit1/cash/capabilities/cashItems/20USD1/value Absolute value of all contents, 0 if mixed. May only be modified in an exchange state if applicable. May be a floating point value to allow for coins and notes which have a value which is not a whole multiple of the currency unit.
storage/unit1/cash/capabilities/cashItems/20USD1/release The release of the cash item. The higher this number is, the newer the release. If zero or not reported, there is only one release of that cash item or the device is not capable of distinguishing different release of the item, for example in a simple cash dispenser. An example of how this can be used is being able to sort different releases of the same denomination note to different storage units to take older notes out of circulation. This value is device, banknote reader and currency dependent, therefore a release number of the same cash item will not necessarily have the same value in different systems and any such usage would be specific to a specific device's configuration. Property value constraints: `minimum: 0`
storage/unit1/cash/configuration Indicates what this storage unit is configured as or is being configured to do - where applicable the supported options can be derived from capabilities. If the service supports an exchange state, only a subset of these parameters may be modified unless in an exchange. Parameters which may only be modified in an exchange state are listed.
storage/unit1/cash/configuration/types The types of operation the unit is capable of or configured to perform. This is a combination of one or more operations. May only be modified in an exchange state if applicable.
storage/unit1/cash/configuration/items The types of cash media the unit is capable of storing or configured to store. This is a combination of one or more item types. May only be modified in an exchange state if applicable. See Note Classification for more information about cash classification levels.
storage/unit1/cash/configuration/currency ISO 4217 currency. May only be modified in an exchange state if applicable. Property value constraints: `maxLength: 3`
storage/unit1/cash/configuration/highThreshold If specified, replenishmentStatus is set to *high* if count is greater than this number. If not specified, high is based on hardware sensors if supported - see hardwareSensors. Property value constraints: `minimum: 1`
storage/unit1/cash/configuration/lowThreshold If specified, replenishmentStatus is set to *low* if count is less than this number. If not specified, low is based on hardware sensors if supported - see hardwareSensors. Property value constraints: `minimum: 1`

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage/unit1/cash/configuration/appLockIn If true, items cannot be accepted into the storage unit in Cash In operations.
storage/unit1/cash/configuration/appLockOut If true, items cannot be dispensed from the storage unit in Cash Out operations.
storage/unit1/cash/configuration/cashItems An object containing multiple cash items, listing what a storage unit is capable of or configured to handle.
storage/unit1/cash/configuration/name Application configured name of the unit.
storage/unit1/cash/configuration/maxRetracts If specified, this is the number of retract operations allowed into the unit. If not specified, the maximum number is not limited by counts. Property value constraints: `minimum: 1`
storage/unit1/cash/status Indicates the storage unit status - this includes information which is a combination of that reported in WFS_INF_CDM_CASH_UNIT_INFO and WFS_INF_CIM_CASH_UNIT_INFO in XFS 3.x.
storage/unit1/cash/status/index Assigned by the XFS4IoT service. Will be a unique number which can be used to determine *usNumber* in XFS 3.x migration. This can change as cash storage units are added and removed from the storage collection. Property value constraints: `minimum: 1`
storage/unit1/cash/status/initial The cash related items which were in the storage unit at the last replenishment.
storage/unit1/cash/status/initial/unrecognized Count of unrecognized items handled by the cash interface
storage/unit1/cash/status/initial/cash Counts of cash items broken down by cash item type and classification
storage/unit1/cash/status/initial/cash/20USD1 (example name) Counts of a given CashItem broken down by classification.
storage/unit1/cash/status/initial/cash/20USD1/fit Count of genuine cash items which are fit for recycling (classification level 4 - see Note Classification).
storage/unit1/cash/status/initial/cash/20USD1/unfit Count of genuine cash items which are unfit for recycling (classification level 4 - see Note Classification).
storage/unit1/cash/status/initial/cash/20USD1/suspect Count of suspect cash items (classification level 3 - see Note Classification).
storage/unit1/cash/status/initial/cash/20USD1/counterfeit Count of counterfeit cash items (classification level 2 - see Note Classification).
storage/unit1/cash/status/initial/cash/20USD1/inked Count of cash items which have been identified as ink stained.
storage/unit1/cash/status/out The items moved from this storage unit by cash commands to another destination since the last replenishment of this unit. Reset to empty if initial is set for this unit by Storage.GetStorage.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage/unit1/cash/status/out/presented The items dispensed from this storage unit which are or were customer accessible.
storage/unit1/cash/status/out/rejected The items dispensed from this storage unit which were invalid and were diverted to a reject storage unit and were not customer accessible during the operation.
storage/unit1/cash/status/out/distributed The items dispensed from this storage unit which were moved to a storage unit other than a reject storage unit and were not customer accessible during the operation.
storage/unit1/cash/status/out/unknown The items dispensed from this storage unit which moved to an unknown position.
storage/unit1/cash/status/out/stacked The items dispensed from this storage unit which are not customer accessible and are currently stacked awaiting presentation to the customer. This item list can increase and decrease as items are moved around in the device.
storage/unit1/cash/status/out/diverted The items dispensed from this storage unit which are not customer accessible and were diverted to a temporary location due to being invalid and have not yet been deposited in a storage unit. This item list can increase and decrease as items are moved around in the device.
storage/unit1/cash/status/out/transport The items dispensed from this storage unit which are not customer accessible and which have jammed in the transport. This item list can increase and decrease as items are moved around in the device.
storage/unit1/cash/status/in List of items inserted in this storage unit by cash commands from another source since the last replenishment of this unit. Reset to empty if initial is set for this unit by Storage.GetStorage.
storage/unit1/cash/status/in/retractOperations Number of cash retract operations which resulted in items entering this storage unit. This can be used where devices do not have the capability to count or validate items after presentation.
storage/unit1/cash/status/in/deposited The items deposited in the storage unit during a Cash In transaction.
storage/unit1/cash/status/in/retracted The items deposited in the storage unit after being accessible to a customer. This may be inaccurate or not counted if items are not counted or re-validated after presentation, the number of retract operations is also reported separately in *retractOperations*.
storage/unit1/cash/status/in/rejected The items deposited in this storage unit originating from another storage unit but rejected due to being invalid. This count may be inaccurate due to the nature of rejected items.
storage/unit1/cash/status/in/distributed The items deposited in this storage unit originating from another storage unit but not rejected.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage/unit1/cash/status/count Total count of the items in the unit, derived from the initial, out and in counts, but supplied for ease of use. For units which dispense items, this count is only decremented when the items are either known to be in customer access or successfully rejected, therefore the intermediate *out* fields are not included in this calculation: • stacked • transport • unknown If counts being incorrectly set at replenishment time means that this would result in a negative number, this reports 0. Property value constraints: `minimum: 0`
storage/unit1/cash/status/accuracy Describes the accuracy of *count*. Following values are possible: • `notSupported` - The hardware is not capable of determining the accuracy of *count*. • `accurate` - The *count* is expected to be accurate. The notes were previously counted and there have since been no events that might have introduced inaccuracy. • `accurateSet` - The *count* is expected to be accurate. The counts 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 storage unit insertion or some other hardware event.
storage/unit1/cash/status/replenishmentStatus The state of the media in the unit if it can be determined. Note that overall status of the storage unit must be taken into account when deciding whether the storage unit is usable and whether replenishment status is applicable. In particular, if the overall status is *missing* this will not be reported. The following values are possible: • `ok` - The storage unit media is in a good state. • `full` - The storage unit is full. • `high` - The storage unit is almost full (either sensor based or exceeded the highThreshold. • `low` - The storage unit is almost empty (either sensor based or below the lowThreshold). • `empty` - The storage unit is empty, or insufficient items in the storage unit are preventing further dispense operations.
storage/unit1/card The card related contents, status and configuration of the unit.
storage/unit1/card/capabilities Indicates the card storage unit capabilities.
storage/unit1/card/capabilities/type The type of card storage as one of the following: • `retain` - The storage unit can retain cards. • `dispense` - The storage unit can dispense cards. • `park` - The storage unit can be used to temporarily store a card allowing another card to enter the transport.
storage/unit1/card/capabilities/hardwareSensors The storage unit has hardware sensors that can detect threshold states. default: false

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage/unit1/card/configuration Indicates the card storage unit configuration.
storage/unit1/card/configuration/cardID The identifier that may be used to identify the type of cards in the storage unit. This is only applicable to *dispense* type storage units.
storage/unit1/card/configuration/threshold If the threshold value is non zero, hardware sensors in the storage unit do not trigger Storage.StorageThresholdEvent events. If non zero, when *count* reaches the threshold value: • For retain type storage units, a high threshold will be sent. • For dispense type storage units, a low threshold will be sent. Property value constraints: `minimum: 0`
storage/unit1/card/status Indicates the card storage unit status.
storage/unit1/card/status/initialCount The initial number of cards in the storage unit. This is only applicable to dispense type storage units. This value is persistent. Property value constraints: `minimum: 0`
storage/unit1/card/status/count The number of cards in the storage unit. If the storage unit type is dispense: • This count also includes a card dispensed from the storage unit which has not been moved to either the exit position or a dispense type storage unit. • This count is decremented when a card from the card storage unit is moved to the exit position or retained. If this value reaches zero it will not decrement further but will remain at zero. If the storage unit type is retain: • The count is incremented when a card is moved into the storage unit. If the storage unit type is park: • The count will increment when a card is moved into the storage module and decremented when a card is moved out of the storage module. This value is persistent. Property value constraints: `minimum: 0`
storage/unit1/card/status/retainCount The number of cards from this storage unit which are in a retain storage unit. This is only applicable to dispense type storage units. This value is persistent. Property value constraints: `minimum: 0`

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage/unit1/card/status/replenishmentStatus The state of the cards in the storage unit if it can be determined. Note that overall status of the storage unit must be taken into account when deciding whether the storage unit is usable and whether replenishment status is applicable. In particular, if the overall status is *missing* this will be omitted. The following values are possible: • `ok` - The storage unit is in a good state. • `full` - The storage unit is full. • `high` - The storage unit is almost full (either sensor based or above the threshold). • `low` - The storage unit is almost empty (either sensor based or below the threshold). • `empty` - The storage unit is empty.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

21.5.2 Storage.StorageThresholdEvent

This user event is generated when a threshold condition has occurred in one of the storage units.

This event can be triggered either by hardware sensors in the device or by count thresholds being met.

Payload	Type	Required

{
  "storage": {
  "unit1": {
  "positionName": "Top Right",
  "capacity": 100,
  "status": "ok",
  "serialNumber": "ABCD1234",
  "cash": {
  "capabilities": {
  "types": {
  "cashIn": false,
  "cashOut": false,
  "replenishment": false,
  "cashInRetract": false,
  "cashOutRetract": false,
  "reject": false
  },
  "items": {
  "fit": false,
  "unfit": false,
  "unrecognized": false,
  "counterfeit": false,
  "suspect": false,
  "inked": false,
  "coupon": false,
  "document": false
  },
  "hardwareSensors": false,
  "retractAreas": 1,
  "retractThresholds": false,
  "cashItems": {
  "20USD1": {
  "noteID": 1,		

	object	
	object	
	string	
	integer	
	string	Yes
	string	
	object	
	object	
	object	
	object	
	boolean	
	boolean	
	boolean	
	boolean	
	boolean	
	boolean	
	object	
	boolean	
	boolean	
	boolean	
	boolean	
	boolean	
	boolean	
	boolean	
	boolean	
	boolean	
	integer	
	boolean	
	object	
	object	
	integer	

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Payload	Type	Required
"currency": "USD",	string	
"value": 20,	number	
"release": 1	integer	
}		
}		
},		
"configuration": {	object	
"types": {	object	
See types properties.		
},		
"items": {	object	
See items properties.		
},		
"currency": "USD",	string	
"value": 20,	number	
"highThreshold": 500,	integer	
"lowThreshold": 10,	integer	
"appLockIn": false,	boolean	
"appLockOut": false,	boolean	
"cashItems": {	object	
See cashItems properties.		
},		
"name": "$10",	string	
"maxRetracts": 5	integer	
},		
"status": {	object	
"index": 1,	integer	
"initial": {	object	
"unrecognized": 0,	integer	
"cash": {	object	
"*20USD1*": {	object	
"fit": 0,	integer	
"unfit": 0,	integer	
"suspect": 0,	integer	
"counterfeit": 0,	integer	
"inked": 0	integer	
}		

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Payload	Type	Required
}		
},		
"out": {	object	
"presented": {	object	
See initial properties.		
},		
"rejected": {	object	
See initial properties.		
},		
"distributed": {	object	
See initial properties.		
},		
"unknown": {	object	
See initial properties.		
},		
"stacked": {	object	
See initial properties.		
},		
"diverted": {	object	
See initial properties.		
},		
"transport": {	object	
See initial properties.		
}		
},		
"in": {	object	
"retractOperations": 0,	integer	
"deposited": {	object	
See initial properties.		
},		
"retracted": {	object	
See initial properties.		
},		
"rejected": {	object	
See initial properties.		
},		
"distributed": {	object	

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Payload	Type	Required
See initial properties.		
}		
},		
"count": 0,	integer	
"accuracy": "notSupported",	string	
"replenishmentStatus": "ok"	string	
}		
},		
"card": {	object	
"capabilities": {	object	
"type": "retain",	string	
"hardwareSensors": false	boolean	
},		
"configuration": {	object	
"cardID": "LoyaltyCard",	string	
"threshold": 0	integer	
},		
"status": {	object	
"initialCount": 0,	integer	
"count": 0,	integer	
"retainCount": 0,	integer	
"replenishmentStatus": "ok"	string	
}		
}		
},		
"unit2": {	object	
See unit1 properties.		
}		
}		
}		

Properties

storage
Object containing storage unit information. The property name is the storage unit identifier.

storage/unit1 (example name)
The object contains a single storage unit.
Property name constraints:
`pattern: ^.{1,5}$`

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage/unit1/positionName Fixed physical name for the position.
storage/unit1/capacity The nominal capacity of the unit. This may be an estimate as the quality and thickness of the items stored in the unit may affect how many items can be stored. 0 means the capacity is unknown. Property value constraints: `minimum: 0`
storage/unit1/status The state of the unit. The following values are possible: • `ok` - The storage unit is in a good state. • `inoperative` - The storage unit is inoperative. • `missing` - The storage unit is missing. • `notConfigured` - The storage unit has not been configured for use. • `manipulated` - The storage unit has been inserted (including removal followed by a reinsertion) when the device was not in the exchange state - see Storage.StartExchange. This storage unit cannot be used. Only applies to services which support the exchange state.
storage/unit1/serialNumber The storage unit's serial number if it can be read electronically.
storage/unit1/cash The cash related contents, status and configuration of the unit.
storage/unit1/cash/capabilities Indicates what the storage unit is capable of - this includes information which is a combination of that reported in WFS_INF_CDM_CASH_UNIT_INFO, WFS_INF_CIM_CASH_UNIT_INFO and WFS_INF_CIM_CASH_UNIT_CAPABILITIES in XFS 3.x.
storage/unit1/cash/capabilities/types The types of operation the unit is capable of or configured to perform. This is a combination of one or more operations. May only be modified in an exchange state if applicable.
storage/unit1/cash/capabilities/types/cashIn The unit can accept cash items. If *cashOut* is also true then the unit can recycle.
storage/unit1/cash/capabilities/types/cashOut The unit can dispense cash items. If *cashIn* is also true then the unit can recycle.
storage/unit1/cash/capabilities/types/replenishment Replenishment container. A storage unit can be refilled from or emptied to a replenishment container.
storage/unit1/cash/capabilities/types/cashInRetract Retract unit. Items can be retracted into this unit during Cash In operations.
storage/unit1/cash/capabilities/types/cashOutRetract Retract unit. Items can be retracted into this unit during Cash Out operations.
storage/unit1/cash/capabilities/types/reject Reject unit. Items can be rejected into this unit.
storage/unit1/cash/capabilities/items The types of cash media the unit is capable of storing or configured to store. This is a combination of one or more item types. May only be modified in an exchange state if applicable. See Note Classification for more information about cash classification levels.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage/unit1/cash/capabilities/items/fit The storage unit can store cash items which are fit for recycling (classification level 4).
storage/unit1/cash/capabilities/items/unfit The storage unit can store cash items which are unfit for recycling (classification level 4).
storage/unit1/cash/capabilities/items/unrecognized The storage unit can store unrecognized cash items (classification level 1).
storage/unit1/cash/capabilities/items/counterfeit The storage unit can store counterfeit cash items (classification level 2).
storage/unit1/cash/capabilities/items/suspect The storage unit can store suspect cash items (classification level 3).
storage/unit1/cash/capabilities/items/inked The storage unit can store cash items which have been identified as ink stained.
storage/unit1/cash/capabilities/items/coupon Storage unit containing coupons or advertising material.
storage/unit1/cash/capabilities/items/document Storage unit containing documents.
storage/unit1/cash/capabilities/hardwareSensors Indicates whether the storage unit has sensors which report the status.
storage/unit1/cash/capabilities/retractAreas If items can be retracted into this storage unit, this is the number of areas within the storage unit which allow physical separation of different bunches. If there is no physical separation of retracted bunches within this storage unit, this value is 1. Property value constraints: `minimum: 1`
storage/unit1/cash/capabilities/retractThresholds If true, indicates that retract capacity is based on counts. If false, indicates that retract capacity is based on the number of commands which resulted in items being retracted into the storage unit.
storage/unit1/cash/capabilities/cashItems Lists the cash items which the storage unit is physically capable of handling. For example a coin storage unit may be restricted to one coin denomination due to the hardware.
storage/unit1/cash/capabilities/cashItems/20USD1 (example name) An object containing information about a single cash item supported by the device.
storage/unit1/cash/capabilities/cashItems/20USD1/noteID Assigned by the XFS4IoT service. A unique number identifying a single cash item. Each unique combination of the other properties will have a different noteID. Can be used for migration of *usNoteID* from XFS 3.x. Property value constraints: `minimum: 1`
storage/unit1/cash/capabilities/cashItems/20USD1/currency ISO 4217 currency. Property value constraints: `maxLength: 3`

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage/unit1/cash/capabilities/cashItems/20USD1/value Absolute value of all contents, 0 if mixed. May only be modified in an exchange state if applicable. May be a floating point value to allow for coins and notes which have a value which is not a whole multiple of the currency unit.
storage/unit1/cash/capabilities/cashItems/20USD1/release The release of the cash item. The higher this number is, the newer the release. If zero or not reported, there is only one release of that cash item or the device is not capable of distinguishing different release of the item, for example in a simple cash dispenser. An example of how this can be used is being able to sort different releases of the same denomination note to different storage units to take older notes out of circulation. This value is device, banknote reader and currency dependent, therefore a release number of the same cash item will not necessarily have the same value in different systems and any such usage would be specific to a specific device's configuration. Property value constraints: `minimum: 0`
storage/unit1/cash/configuration Indicates what this storage unit is configured as or is being configured to do - where applicable the supported options can be derived from capabilities. If the service supports an exchange state, only a subset of these parameters may be modified unless in an exchange. Parameters which may only be modified in an exchange state are listed.
storage/unit1/cash/configuration/types The types of operation the unit is capable of or configured to perform. This is a combination of one or more operations. May only be modified in an exchange state if applicable.
storage/unit1/cash/configuration/items The types of cash media the unit is capable of storing or configured to store. This is a combination of one or more item types. May only be modified in an exchange state if applicable. See Note Classification for more information about cash classification levels.
storage/unit1/cash/configuration/currency ISO 4217 currency. May only be modified in an exchange state if applicable. Property value constraints: `maxLength: 3`
storage/unit1/cash/configuration/highThreshold If specified, replenishmentStatus is set to *high* if count is greater than this number. If not specified, high is based on hardware sensors if supported - see hardwareSensors. Property value constraints: `minimum: 1`
storage/unit1/cash/configuration/lowThreshold If specified, replenishmentStatus is set to *low* if count is less than this number. If not specified, low is based on hardware sensors if supported - see hardwareSensors. Property value constraints: `minimum: 1`
storage/unit1/cash/configuration/appLockIn If true, items cannot be accepted into the storage unit in Cash In operations.
storage/unit1/cash/configuration/appLockOut If true, items cannot be dispensed from the storage unit in Cash Out operations.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage/unit1/cash/configuration/cashItems An object containing multiple cash items, listing what a storage unit is capable of or configured to handle.
storage/unit1/cash/configuration/name Application configured name of the unit.
storage/unit1/cash/configuration/maxRetracts If specified, this is the number of retract operations allowed into the unit. If not specified, the maximum number is not limited by counts. Property value constraints: `minimum: 1`
storage/unit1/cash/status Indicates the storage unit status - this includes information which is a combination of that reported in WFS_INF_CDM_CASH_UNIT_INFO and WFS_INF_CIM_CASH_UNIT_INFO in XFS 3.x.
storage/unit1/cash/status/index Assigned by the XFS4IoT service. Will be a unique number which can be used to determine *usNumber* in XFS 3.x migration. This can change as cash storage units are added and removed from the storage collection. Property value constraints: `minimum: 1`
storage/unit1/cash/status/initial The cash related items which were in the storage unit at the last replenishment.
storage/unit1/cash/status/initial/unrecognized Count of unrecognized items handled by the cash interface
storage/unit1/cash/status/initial/cash Counts of cash items broken down by cash item type and classification
storage/unit1/cash/status/initial/cash/20USD1 (example name) Counts of a given CashItem broken down by classification.
storage/unit1/cash/status/initial/cash/20USD1/fit Count of genuine cash items which are fit for recycling (classification level 4 - see Note Classification).
storage/unit1/cash/status/initial/cash/20USD1/unfit Count of genuine cash items which are unfit for recycling (classification level 4 - see Note Classification).
storage/unit1/cash/status/initial/cash/20USD1/suspect Count of suspect cash items (classification level 3 - see Note Classification).
storage/unit1/cash/status/initial/cash/20USD1/counterfeit Count of counterfeit cash items (classification level 2 - see Note Classification).
storage/unit1/cash/status/initial/cash/20USD1/inked Count of cash items which have been identified as ink stained.
storage/unit1/cash/status/out The items moved from this storage unit by cash commands to another destination since the last replenishment of this unit. Reset to empty if initial is set for this unit by Storage.GetStorage.
storage/unit1/cash/status/out/presented The items dispensed from this storage unit which are or were customer accessible.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage/unit1/cash/status/out/rejected The items dispensed from this storage unit which were invalid and were diverted to a reject storage unit and were not customer accessible during the operation.
storage/unit1/cash/status/out/distributed The items dispensed from this storage unit which were moved to a storage unit other than a reject storage unit and were not customer accessible during the operation.
storage/unit1/cash/status/out/unknown The items dispensed from this storage unit which moved to an unknown position.
storage/unit1/cash/status/out/stacked The items dispensed from this storage unit which are not customer accessible and are currently stacked awaiting presentation to the customer. This item list can increase and decrease as items are moved around in the device.
storage/unit1/cash/status/out/diverted The items dispensed from this storage unit which are not customer accessible and were diverted to a temporary location due to being invalid and have not yet been deposited in a storage unit. This item list can increase and decrease as items are moved around in the device.
storage/unit1/cash/status/out/transport The items dispensed from this storage unit which are not customer accessible and which have jammed in the transport. This item list can increase and decrease as items are moved around in the device.
storage/unit1/cash/status/in List of items inserted in this storage unit by cash commands from another source since the last replenishment of this unit. Reset to empty if initial is set for this unit by Storage.GetStorage.
storage/unit1/cash/status/in/retractOperations Number of cash retract operations which resulted in items entering this storage unit. This can be used where devices do not have the capability to count or validate items after presentation.
storage/unit1/cash/status/in/deposited The items deposited in the storage unit during a Cash In transaction.
storage/unit1/cash/status/in/retracted The items deposited in the storage unit after being accessible to a customer. This may be inaccurate or not counted if items are not counted or re-validated after presentation, the number of retract operations is also reported separately in *retractOperations*.
storage/unit1/cash/status/in/rejected The items deposited in this storage unit originating from another storage unit but rejected due to being invalid. This count may be inaccurate due to the nature of rejected items.
storage/unit1/cash/status/in/distributed The items deposited in this storage unit originating from another storage unit but not rejected.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage/unit1/cash/status/count Total count of the items in the unit, derived from the initial, out and in counts, but supplied for ease of use. For units which dispense items, this count is only decremented when the items are either known to be in customer access or successfully rejected, therefore the intermediate *out* fields are not included in this calculation: • stacked • transport • unknown If counts being incorrectly set at replenishment time means that this would result in a negative number, this reports 0. Property value constraints: `minimum: 0`
storage/unit1/cash/status/accuracy Describes the accuracy of *count*. Following values are possible: • `notSupported` - The hardware is not capable of determining the accuracy of *count*. • `accurate` - The *count* is expected to be accurate. The notes were previously counted and there have since been no events that might have introduced inaccuracy. • `accurateSet` - The *count* is expected to be accurate. The counts 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 storage unit insertion or some other hardware event.
storage/unit1/cash/status/replenishmentStatus The state of the media in the unit if it can be determined. Note that overall status of the storage unit must be taken into account when deciding whether the storage unit is usable and whether replenishment status is applicable. In particular, if the overall status is *missing* this will not be reported. The following values are possible: • `ok` - The storage unit media is in a good state. • `full` - The storage unit is full. • `high` - The storage unit is almost full (either sensor based or exceeded the highThreshold. • `low` - The storage unit is almost empty (either sensor based or below the lowThreshold). • `empty` - The storage unit is empty, or insufficient items in the storage unit are preventing further dispense operations.
storage/unit1/card The card related contents, status and configuration of the unit.
storage/unit1/card/capabilities Indicates the card storage unit capabilities.
storage/unit1/card/capabilities/type The type of card storage as one of the following: • `retain` - The storage unit can retain cards. • `dispense` - The storage unit can dispense cards. • `park` - The storage unit can be used to temporarily store a card allowing another card to enter the transport.
storage/unit1/card/capabilities/hardwareSensors The storage unit has hardware sensors that can detect threshold states. default: false

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage/unit1/card/configuration Indicates the card storage unit configuration.
storage/unit1/card/configuration/cardID The identifier that may be used to identify the type of cards in the storage unit. This is only applicable to *dispense* type storage units.
storage/unit1/card/configuration/threshold If the threshold value is non zero, hardware sensors in the storage unit do not trigger Storage.StorageThresholdEvent events. If non zero, when *count* reaches the threshold value: • For retain type storage units, a high threshold will be sent. • For dispense type storage units, a low threshold will be sent. Property value constraints: `minimum: 0`
storage/unit1/card/status Indicates the card storage unit status.
storage/unit1/card/status/initialCount The initial number of cards in the storage unit. This is only applicable to dispense type storage units. This value is persistent. Property value constraints: `minimum: 0`
storage/unit1/card/status/count The number of cards in the storage unit. If the storage unit type is dispense: • This count also includes a card dispensed from the storage unit which has not been moved to either the exit position or a dispense type storage unit. • This count is decremented when a card from the card storage unit is moved to the exit position or retained. If this value reaches zero it will not decrement further but will remain at zero. If the storage unit type is retain: • The count is incremented when a card is moved into the storage unit. If the storage unit type is park: • The count will increment when a card is moved into the storage module and decremented when a card is moved out of the storage module. This value is persistent. Property value constraints: `minimum: 0`
storage/unit1/card/status/retainCount The number of cards from this storage unit which are in a retain storage unit. This is only applicable to dispense type storage units. This value is persistent. Property value constraints: `minimum: 0`

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
storage/unit1/card/status/replenishmentStatus The state of the cards in the storage unit if it can be determined. Note that overall status of the storage unit must be taken into account when deciding whether the storage unit is usable and whether replenishment status is applicable. In particular, if the overall status is *missing* this will be omitted. The following values are possible: • `ok` - The storage unit is in a good state. • `full` - The storage unit is full. • `high` - The storage unit is almost full (either sensor based or above the threshold). • `low` - The storage unit is almost empty (either sensor based or below the threshold). • `empty` - The storage unit is empty.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

21.5.3 Storage.StorageErrorEvent

This event is generated if there is a problem with a storage unit during the execution of a command.

Payload	Type	Required
{		
"failure": "empty",	string	
"unit": {	object	
"storage": {	object	
"unit1": {	object	
"positionName": "Top Right",	string	
"capacity": 100,	integer	
"status": "ok",	string	Yes
"serialNumber": "ABCD1234",	string	
"cash": {	object	
"capabilities": {	object	
"types": {	object	
"cashIn": false,	boolean	
"cashOut": false,	boolean	
"replenishment": false,	boolean	
"cashInRetract": false,	boolean	
"cashOutRetract": false,	boolean	
"reject": false	boolean	
},		
"items": {	object	
"fit": false,	boolean	
"unfit": false,	boolean	
"unrecognized": false,	boolean	
"counterfeit": false,	boolean	
"suspect": false,	boolean	
"inked": false,	boolean	
"coupon": false,	boolean	
"document": false	boolean	
},		
"hardwareSensors": false,	boolean	
"retractAreas": 1,	integer	
"retractThresholds": false,	boolean	
"cashItems": {	object	
"20USD1": {	object	

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Payload	Type	Required
"noteID": 1,	integer	
"currency": "USD",	string	
"value": 20,	number	
"release": 1	integer	
}		
}		
},		
"configuration": {	object	
"types": {	object	
See types properties.		
},		
"items": {	object	
See items properties.		
},		
"currency": "USD",	string	
"value": 20,	number	
"highThreshold": 500,	integer	
"lowThreshold": 10,	integer	
"appLockIn": false,	boolean	
"appLockOut": false,	boolean	
"cashItems": {	object	
See cashItems properties.		
},		
"name": "$10",	string	
"maxRetracts": 5	integer	
},		
"status": {	object	
"index": 1,	integer	
"initial": {	object	
"unrecognized": 0,	integer	
"cash": {	object	
"*20USD1*": {	object	
"fit": 0,	integer	
"unfit": 0,	integer	
"suspect": 0,	integer	
"counterfeit": 0,	integer	
"inked": 0	integer	

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Payload	Type	Required
`}`		
`}`		
`},`		
`"out": {`	object	
`"presented": {`	object	
`See initial properties.`		
`},`		
`"rejected": {`	object	
`See initial properties.`		
`},`		
`"distributed": {`	object	
`See initial properties.`		
`},`		
`"unknown": {`	object	
`See initial properties.`		
`},`		
`"stacked": {`	object	
`See initial properties.`		
`},`		
`"diverted": {`	object	
`See initial properties.`		
`},`		
`"transport": {`	object	
`See initial properties.`		
`}`		
`},`		
`"in": {`	object	
`"retractOperations": 0,`	integer	
`"deposited": {`	object	
`See initial properties.`		
`},`		
`"retracted": {`	object	
`See initial properties.`		
`},`		
`"rejected": {`	object	
`See initial properties.`		
`},`		

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Payload	Type	Required
"distributed": {	object	
See initial properties.		
}		
},		
"count": 0,	integer	
"accuracy": "notSupported",	string	
"replenishmentStatus": "ok"	string	
}		
},		
"card": {	object	
"capabilities": {	object	
"type": "retain",	string	
"hardwareSensors": false	boolean	
},		
"configuration": {	object	
"cardID": "LoyaltyCard",	string	
"threshold": 0	integer	
},		
"status": {	object	
"initialCount": 0,	integer	
"count": 0,	integer	
"retainCount": 0,	integer	
"replenishmentStatus": "ok"	string	
}		
}		
},		
"unit2": {	object	
See unit1 properties.		
}		
}		
}		
}		

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
failure Specifies the kind of failure that occurred in the storage unit. Following values are possible: • `empty` - Specified storage unit is empty. • `error` - Specified storage unit has malfunctioned. • `full` - Specified storage unit is full. • `locked` - Specified storage unit is locked. • `invalid` - Specified storage unit is invalid. • `config` - An attempt has been made to change the settings of a self-configuring storage unit. • `notConfigured` - Specified storage unit is not configured.
unit The storage unit object that caused the problem.
unit/storage Object containing storage unit information. The property name is the storage unit identifier.
unit/storage/unit1 (example name) The object contains a single storage unit. Property name constraints: `pattern: ^.{1,5}$`
unit/storage/unit1/positionName Fixed physical name for the position.
unit/storage/unit1/capacity The nominal capacity of the unit. This may be an estimate as the quality and thickness of the items stored in the unit may affect how many items can be stored. 0 means the capacity is unknown. Property value constraints: `minimum: 0`
unit/storage/unit1/status The state of the unit. The following values are possible: • `ok` - The storage unit is in a good state. • `inoperative` - The storage unit is inoperative. • `missing` - The storage unit is missing. • `notConfigured` - The storage unit has not been configured for use. • `manipulated` - The storage unit has been inserted (including removal followed by a reinsertion) when the device was not in the exchange state - see Storage.StartExchange. This storage unit cannot be used. Only applies to services which support the exchange state.
unit/storage/unit1/serialNumber The storage unit's serial number if it can be read electronically.
unit/storage/unit1/cash The cash related contents, status and configuration of the unit.
unit/storage/unit1/cash/capabilities Indicates what the storage unit is capable of - this includes information which is a combination of that reported in WFS_INF_CDM_CASH_UNIT_INFO, WFS_INF_CIM_CASH_UNIT_INFO and WFS_INF_CIM_CASH_UNIT_CAPABILITIES in XFS 3.x.
unit/storage/unit1/cash/capabilities/types The types of operation the unit is capable of or configured to perform. This is a combination of one or more operations. May only be modified in an exchange state if applicable.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
unit/storage/unit1/cash/capabilities/types/cashIn The unit can accept cash items. If *cashOut* is also true then the unit can recycle.
unit/storage/unit1/cash/capabilities/types/cashOut The unit can dispense cash items. If *cashIn* is also true then the unit can recycle.
unit/storage/unit1/cash/capabilities/types/replenishment Replenishment container. A storage unit can be refilled from or emptied to a replenishment container.
unit/storage/unit1/cash/capabilities/types/cashInRetract Retract unit. Items can be retracted into this unit during Cash In operations.
unit/storage/unit1/cash/capabilities/types/cashOutRetract Retract unit. Items can be retracted into this unit during Cash Out operations.
unit/storage/unit1/cash/capabilities/types/reject Reject unit. Items can be rejected into this unit.
unit/storage/unit1/cash/capabilities/items The types of cash media the unit is capable of storing or configured to store. This is a combination of one or more item types. May only be modified in an exchange state if applicable. See Note Classification for more information about cash classification levels.
unit/storage/unit1/cash/capabilities/items/fit The storage unit can store cash items which are fit for recycling (classification level 4).
unit/storage/unit1/cash/capabilities/items/unfit The storage unit can store cash items which are unfit for recycling (classification level 4).
unit/storage/unit1/cash/capabilities/items/unrecognized The storage unit can store unrecognized cash items (classification level 1).
unit/storage/unit1/cash/capabilities/items/counterfeit The storage unit can store counterfeit cash items (classification level 2).
unit/storage/unit1/cash/capabilities/items/suspect The storage unit can store suspect cash items (classification level 3).
unit/storage/unit1/cash/capabilities/items/inked The storage unit can store cash items which have been identified as ink stained.
unit/storage/unit1/cash/capabilities/items/coupon Storage unit containing coupons or advertising material.
unit/storage/unit1/cash/capabilities/items/document Storage unit containing documents.
unit/storage/unit1/cash/capabilities/hardwareSensors Indicates whether the storage unit has sensors which report the status.
unit/storage/unit1/cash/capabilities/retractAreas If items can be retracted into this storage unit, this is the number of areas within the storage unit which allow physical separation of different bunches. If there is no physical separation of retracted bunches within this storage unit, this value is 1. Property value constraints: `minimum: 1`

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
unit/storage/unit1/cash/capabilities/retractThresholds If true, indicates that retract capacity is based on counts. If false, indicates that retract capacity is based on the number of commands which resulted in items being retracted into the storage unit.
unit/storage/unit1/cash/capabilities/cashItems Lists the cash items which the storage unit is physically capable of handling. For example a coin storage unit may be restricted to one coin denomination due to the hardware.
unit/storage/unit1/cash/capabilities/cashItems/20USD1 (example name) An object containing information about a single cash item supported by the device.
unit/storage/unit1/cash/capabilities/cashItems/20USD1/noteID Assigned by the XFS4IoT service. A unique number identifying a single cash item. Each unique combination of the other properties will have a different noteID. Can be used for migration of *usNoteID* from XFS 3.x. Property value constraints: `minimum: 1`
unit/storage/unit1/cash/capabilities/cashItems/20USD1/currency ISO 4217 currency. Property value constraints: `maxLength: 3`
unit/storage/unit1/cash/capabilities/cashItems/20USD1/value Absolute value of all contents, 0 if mixed. May only be modified in an exchange state if applicable. May be a floating point value to allow for coins and notes which have a value which is not a whole multiple of the currency unit.
unit/storage/unit1/cash/capabilities/cashItems/20USD1/release The release of the cash item. The higher this number is, the newer the release. If zero or not reported, there is only one release of that cash item or the device is not capable of distinguishing different release of the item, for example in a simple cash dispenser. An example of how this can be used is being able to sort different releases of the same denomination note to different storage units to take older notes out of circulation. This value is device, banknote reader and currency dependent, therefore a release number of the same cash item will not necessarily have the same value in different systems and any such usage would be specific to a specific device's configuration. Property value constraints: `minimum: 0`
unit/storage/unit1/cash/configuration Indicates what this storage unit is configured as or is being configured to do - where applicable the supported options can be derived from capabilities. If the service supports an exchange state, only a subset of these parameters may be modified unless in an exchange. Parameters which may only be modified in an exchange state are listed.
unit/storage/unit1/cash/configuration/types The types of operation the unit is capable of or configured to perform. This is a combination of one or more operations. May only be modified in an exchange state if applicable.
unit/storage/unit1/cash/configuration/items The types of cash media the unit is capable of storing or configured to store. This is a combination of one or more item types. May only be modified in an exchange state if applicable. See Note Classification for more information about cash classification levels.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
unit/storage/unit1/cash/configuration/currency ISO 4217 currency. May only be modified in an exchange state if applicable. Property value constraints: `maxLength: 3`
unit/storage/unit1/cash/configuration/highThreshold If specified, replenishmentStatus is set to *high* if count is greater than this number. If not specified, high is based on hardware sensors if supported - see hardwareSensors. Property value constraints: `minimum: 1`
unit/storage/unit1/cash/configuration/lowThreshold If specified, replenishmentStatus is set to *low* if count is less than this number. If not specified, low is based on hardware sensors if supported - see hardwareSensors. Property value constraints: `minimum: 1`
unit/storage/unit1/cash/configuration/appLockIn If true, items cannot be accepted into the storage unit in Cash In operations.
unit/storage/unit1/cash/configuration/appLockOut If true, items cannot be dispensed from the storage unit in Cash Out operations.
unit/storage/unit1/cash/configuration/cashItems An object containing multiple cash items, listing what a storage unit is capable of or configured to handle.
unit/storage/unit1/cash/configuration/name Application configured name of the unit.
unit/storage/unit1/cash/configuration/maxRetracts If specified, this is the number of retract operations allowed into the unit. If not specified, the maximum number is not limited by counts. Property value constraints: `minimum: 1`
unit/storage/unit1/cash/status Indicates the storage unit status - this includes information which is a combination of that reported in WFS_INF_CDM_CASH_UNIT_INFO and WFS_INF_CIM_CASH_UNIT_INFO in XFS 3.x.
unit/storage/unit1/cash/status/index Assigned by the XFS4IoT service. Will be a unique number which can be used to determine *usNumber* in XFS 3.x migration. This can change as cash storage units are added and removed from the storage collection. Property value constraints: `minimum: 1`
unit/storage/unit1/cash/status/initial The cash related items which were in the storage unit at the last replenishment.
unit/storage/unit1/cash/status/initial/unrecognized Count of unrecognized items handled by the cash interface
unit/storage/unit1/cash/status/initial/cash Counts of cash items broken down by cash item type and classification

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
unit/storage/unit1/cash/status/initial/cash/20USD1 (example name) Counts of a given CashItem broken down by classification.
unit/storage/unit1/cash/status/initial/cash/20USD1/fit Count of genuine cash items which are fit for recycling (classification level 4 - see Note Classification).
unit/storage/unit1/cash/status/initial/cash/20USD1/unfit Count of genuine cash items which are unfit for recycling (classification level 4 - see Note Classification).
unit/storage/unit1/cash/status/initial/cash/20USD1/suspect Count of suspect cash items (classification level 3 - see Note Classification).
unit/storage/unit1/cash/status/initial/cash/20USD1/counterfeit Count of counterfeit cash items (classification level 2 - see Note Classification).
unit/storage/unit1/cash/status/initial/cash/20USD1/inked Count of cash items which have been identified as ink stained.
unit/storage/unit1/cash/status/out The items moved from this storage unit by cash commands to another destination since the last replenishment of this unit. Reset to empty if initial is set for this unit by Storage.GetStorage.
unit/storage/unit1/cash/status/out/presented The items dispensed from this storage unit which are or were customer accessible.
unit/storage/unit1/cash/status/out/rejected The items dispensed from this storage unit which were invalid and were diverted to a reject storage unit and were not customer accessible during the operation.
unit/storage/unit1/cash/status/out/distributed The items dispensed from this storage unit which were moved to a storage unit other than a reject storage unit and were not customer accessible during the operation.
unit/storage/unit1/cash/status/out/unknown The items dispensed from this storage unit which moved to an unknown position.
unit/storage/unit1/cash/status/out/stacked The items dispensed from this storage unit which are not customer accessible and are currently stacked awaiting presentation to the customer. This item list can increase and decrease as items are moved around in the device.
unit/storage/unit1/cash/status/out/diverted The items dispensed from this storage unit which are not customer accessible and were diverted to a temporary location due to being invalid and have not yet been deposited in a storage unit. This item list can increase and decrease as items are moved around in the device.
unit/storage/unit1/cash/status/out/transport The items dispensed from this storage unit which are not customer accessible and which have jammed in the transport. This item list can increase and decrease as items are moved around in the device.
unit/storage/unit1/cash/status/in List of items inserted in this storage unit by cash commands from another source since the last replenishment of this unit. Reset to empty if initial is set for this unit by Storage.GetStorage.
unit/storage/unit1/cash/status/in/retractOperations Number of cash retract operations which resulted in items entering this storage unit. This can be used where devices do not have the capability to count or validate items after presentation.
unit/storage/unit1/cash/status/in/deposited The items deposited in the storage unit during a Cash In transaction.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
unit/storage/unit1/cash/status/in/retracted The items deposited in the storage unit after being accessible to a customer. This may be inaccurate or not counted if items are not counted or re-validated after presentation, the number of retract operations is also reported separately in *retractOperations*.
unit/storage/unit1/cash/status/in/rejected The items deposited in this storage unit originating from another storage unit but rejected due to being invalid. This count may be inaccurate due to the nature of rejected items.
unit/storage/unit1/cash/status/in/distributed The items deposited in this storage unit originating from another storage unit but not rejected.
unit/storage/unit1/cash/status/count Total count of the items in the unit, derived from the initial, out and in counts, but supplied for ease of use. For units which dispense items, this count is only decremented when the items are either known to be in customer access or successfully rejected, therefore the intermediate *out* fields are not included in this calculation: • stacked • transport • unknown If counts being incorrectly set at replenishment time means that this would result in a negative number, this reports 0. Property value constraints: `minimum: 0`
unit/storage/unit1/cash/status/accuracy Describes the accuracy of *count*. Following values are possible: • `notSupported` - The hardware is not capable of determining the accuracy of *count*. • `accurate` - The *count* is expected to be accurate. The notes were previously counted and there have since been no events that might have introduced inaccuracy. • `accurateSet` - The *count* is expected to be accurate. The counts 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 storage unit insertion or some other hardware event.
unit/storage/unit1/cash/status/replenishmentStatus The state of the media in the unit if it can be determined. Note that overall status of the storage unit must be taken into account when deciding whether the storage unit is usable and whether replenishment status is applicable. In particular, if the overall status is *missing* this will not be reported. The following values are possible: • `ok` - The storage unit media is in a good state. • `full` - The storage unit is full. • `high` - The storage unit is almost full (either sensor based or exceeded the highThreshold. • `low` - The storage unit is almost empty (either sensor based or below the lowThreshold). • `empty` - The storage unit is empty, or insufficient items in the storage unit are preventing further dispense operations.
unit/storage/unit1/card The card related contents, status and configuration of the unit.
unit/storage/unit1/card/capabilities Indicates the card storage unit capabilities.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
unit/storage/unit1/card/capabilities/type The type of card storage as one of the following: • `retain` - The storage unit can retain cards. • `dispense` - The storage unit can dispense cards. • `park` - The storage unit can be used to temporarily store a card allowing another card to enter the transport.
unit/storage/unit1/card/capabilities/hardwareSensors The storage unit has hardware sensors that can detect threshold states. default: false
unit/storage/unit1/card/configuration Indicates the card storage unit configuration.
unit/storage/unit1/card/configuration/cardID The identifier that may be used to identify the type of cards in the storage unit. This is only applicable to *dispense* type storage units.
unit/storage/unit1/card/configuration/threshold If the threshold value is non zero, hardware sensors in the storage unit do not trigger Storage.StorageThresholdEvent events. If non zero, when *count* reaches the threshold value: • For retain type storage units, a high threshold will be sent. • For dispense type storage units, a low threshold will be sent. Property value constraints: `minimum: 0`
unit/storage/unit1/card/status Indicates the card storage unit status.
unit/storage/unit1/card/status/initialCount The initial number of cards in the storage unit. This is only applicable to dispense type storage units. This value is persistent. Property value constraints: `minimum: 0`
unit/storage/unit1/card/status/count The number of cards in the storage unit. If the storage unit type is dispense: • This count also includes a card dispensed from the storage unit which has not been moved to either the exit position or a dispense type storage unit. • This count is decremented when a card from the card storage unit is moved to the exit position or retained. If this value reaches zero it will not decrement further but will remain at zero. If the storage unit type is retain: • The count is incremented when a card is moved into the storage unit. If the storage unit type is park: • The count will increment when a card is moved into the storage module and decremented when a card is moved out of the storage module. This value is persistent. Property value constraints: `minimum: 0`

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
unit/storage/unit1/card/status/retainCount The number of cards from this storage unit which are in a retain storage unit. This is only applicable to dispense type storage units. This value is persistent. Property value constraints: `minimum: 0`
unit/storage/unit1/card/status/replenishmentStatus The state of the cards in the storage unit if it can be determined. Note that overall status of the storage unit must be taken into account when deciding whether the storage unit is usable and whether replenishment status is applicable. In particular, if the overall status is *missing* this will be omitted. The following values are possible: • `ok` - The storage unit is in a good state. • `full` - The storage unit is full. • `high` - The storage unit is almost full (either sensor based or above the threshold). • `low` - The storage unit is almost empty (either sensor based or below the threshold). • `empty` - The storage unit is empty.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

22. Vendor Mode Interface

This chapter defines the VDM interface functionality and messages.

22.1 Summary

This interface allows for the coordination of access to resources, and should be read in conjunction with the Vendor Application interface.

22.2 General Information

22.2.1 Vendor Mode

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

23. Vendor Mode

This specification describes the functionality of the services provided by the Vendor Mode Service under XFS4IoT, by defining the service-specific commands that can be issued.

In all device classes there needs to be some method of going into a vendor specific mode to allow for capabilities which go beyond the scope of the current XFS4IoT specifications. A typical usage of such a mode might be to handle some configuration or diagnostic type of function or perhaps perform some 'off-line' testing of the device. These functions are normally available on Self-Service devices in a mode traditionally referred to as Maintenance Mode or Supervisor Mode and usually require operator intervention. It is those vendor-specific functions not covered by (and not required to be covered by) XFS4IoT Services that will be available once the device is in Vendor Mode. This service provides the mechanism for switching to and from Vendor Mode. The Vendor Mode service can be seen as the central point through which all Enter and Exit Vendor Mode requests are synchronized. Entry into, or exit from, Vendor Mode can be initiated either by an application or by the Vendor Mode Service Provider itself. If initiated by an application, then this application needs to issue the appropriate command to request entry or exit. If initiated by the Vendor Mode service i.e. some vendor dependent switch, then these request commands are inappropriate and not issued. Once the entry request has been made, all registered applications will be notified of the entry request by an event message. These applications must attempt to close all open sessions with XFS Service Providers (except for specific Services which explicitly allow sessions to remain open) as soon as possible and then issue an acknowledgement command to the Vendor Mode service when ready. Once all applications have acknowledged, the Vendor Mode Service Provider will issue event messages to these applications to indicate that the System is in Vendor Mode. The application can then start the vendor dependent application.

Similarly, once the exit request has been made all registered applications will be notified of the exit request by an event message. These applications must then issue an acknowledgement command to the Vendor Mode service immediately. Once all applications have acknowledged, the Vendor Mode service will issue event messages to these applications to indicate that the system has exited from Vendor Mode.

The Vendor Mode service is used in conjunction with the Vendor Application service. The VDM service is responsible for coordinating the release of resources from other services, while the Vendor Application service is responsible for starting the vendor dependent application. The VendorApplication.StartLocalApplication command is used for the latter.

With regard to how the application relates to other services the follow two rules apply:

1. If the vendor dependent application is published on the same service as a device, then the app only applies to that service/device.
2. If the vendor dependent application is on its own service without any other device classes, then the app applies to all services/devices published by that publisher - i.e. from that vendor/hardware manufacturer.

The Following diagrams show the various methods of entering and exiting Vendor Mode and should be read in conjunction with the command and event descriptions:

23.1 Vendor Mode Entry triggered by an XFS Application

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

1. An XFS4IoT application calls the VendorMode.EnterModeRequest Command on the Vendor Mode Service to request entry into Vendor Mode.
2. The Vendor Mode service sends an VendorMode.EnterModeRequestEvent to all applications which have registered to participate in Vendor Mode. The applications relinquish control of the services they are connected to when and if they can.
3. Once the other applications have relinquished control of their device resources they send an VendorMode.EnterModeAcknowledge event. command to the Vendor Mode service.
4. When all registered applications have reported that they have relinquished control of their services the Vendor Mode service sends a VendorMode.ModeEnteredEvent to signify entry into Vendor Mode.
5. The Application calls the VendorApplication.StartLocalApplication command to start the vendor dependent application.

The system is now in Vendor Mode and a vendor dependent application can exclusively use the system devices in a vendor dependent manner.

23.2 Vendor Mode Exit

1. The vendor dependent application exits.
2. The XFS4IoT application calls the VendorMode.ExitModeRequest command on the service to request exit from Vendor Mode.
3. The Vendor Mode Service sends an VendorMode.ExitModeRequestEvent event to all applications which have registered to participate in Vendor Mode.
4. The other applications call the VendorMode.ExitModeAcknowledge command on the Vendor Mode service.
5. When all registered applications have reported that they have exited Vendor Mode the service sends a VendorMode.ModeExitedEvent event to report exit from Vendor Mode.

The system is no longer in Vendor Mode.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

23.3 Command Messages

23.3.1 VendorMode.Register

This command is issued by an application to register that it wants to participate in Vendor Mode.

Command Message

Payload	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"appName": 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 canceled.
default: 0

appName
Specifies a logical name for the application that is registering. It should give some indication of the identity and function of the registering application, e.g. "NCR Monitoring app".

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

23.3.2 VendorMode.EnterModeRequest

This command is issued by an application to indicate a logical request to enter Vendor Mode. Theservice will then indicate the request to all registered applications by sending a VendorMode.EnterModeRequestEvent and then wait for an acknowledgement back from each registered application before putting the system into Vendor Mode. The service status will change to enterPending on receipt of this command and will prevail until all applications have acknowledged, at which time the status will change to active and the EnterModeRequest completes. If the command fails when the status is enterPending, the status is changed to inactive and a VendorMode.ModeExitedEvent is sent to all registered applications.

Command Message

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	string	
`}`		

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

23.3.3 VendorMode.EnterModeAcknowledge

This command is issued by a registered application as an acknowledgement to the VendorMode.EnterModeRequestEvent and it indicates that it is ready for the system to enter Vendor Mode. All registered applications must respond before Vendor Mode can be entered. Completion of this command is immediate. If this command is executed outwith a request for VDM entry, or if the acknowledge has already been sent from this connection then the command completes with a sequenceError error code.

Note: Applications must be prepared to allow the vendor dependent application to display on the active interface. This means that applications should no longer try to be the foreground or topmost window to ensure that the vendor dependent application is visible.

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 canceled.
default: 0

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

23.3.4 VendorMode.ExitModeRequest

This command is issued by an application to indicate a request to exit Vendor Mode. The Service will then indicate the request to all registered applications by sending a VendorMode.ExitModeRequestEvent and then wait for an acknowledgement back from each registered application before removing the system from Vendor Mode. The status will change to exitPending on receipt of this command and will prevail until all applications have acknowledged, at which time the status will change to inactive and the ExitModeRequest command completes. If the command fails when the status is exitPending, the status is changed to active and a VendorMode.ModeEnteredEvent is sent to all registered applications.

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 canceled.
default: 0

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

- VendorMode.ExitModeRequestEvent
- VendorMode.ModeEnteredEvent
- VendorMode.ModeExitedEvent

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

23.3.5 VendorMode.ExitModeAcknowledge

This command is issued by a registered application as an acknowledgement to the VendorMode.ExitModeRequest command and it indicates that the application is ready for the system to exit Vendor Mode. All registered applications (including the application that issued the request to exit Vendor Mode) must respond before Vendor Mode will be exited. Completion of this command is immediate.

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 canceled.
default: 0

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

23.4 Unsolicited Messages

23.4.1 VendorMode.EnterModeRequestEvent

This service event is used to indicate the request to enter Vendor Dependent Mode.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

23.4.2 VendorMode.ExitModeRequestEvent

This service event is used to indicate the request to exit Vendor Dependent Mode.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

23.4.3 VendorMode.ModeEnteredEvent

This event is used to indicate that the system has entered Vendor Dependent Mode.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

23.4.4 VendorMode.ModeExitedEvent

This event is used to indicate that the system has exited Vendor Dependent Mode.

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"connectedApplications": [Add example to YAML]`	array (string)	
`}`		

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
connectedApplications List of applications that have not shut down.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

24. Vendor Application Interface

This chapter defines the Vendor Application Service.

24.1 Summary

This specification describes the functionality of the Vendor Application Service, which is used to start a local application, and set the active interface.

24.2 General Information

24.2.1 Vendor Application

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

25. Vendor Application

This specification describes the functionality of the commands and events provided by the Vendor Application Service by defining the service-specific commands that can be used. This service is responsible for starting a local vendor dependent application and should be used in conjunction with the VendorMode service, which is responsible for managing arbitration of access to active services in the services environment. For the exact detail of the interaction between the VendorMode service and the StartLocalApplication service refer to the Vendor Mode services documentation, which describes this fully.

The VendorMode service is solely responsible for allowing an application to inform devices to either relinquish or reassert control of hardware, whereas the VendorApplication service is responsible for starting and managing the vendor dependent application itself. The vendor dependent application could be a monitoring application, a maintenance application or have another purpose. The exact purpose is not mandated by XFS4IoT.

Once the VendorMode interface has been called the StartLocalApplication command can be used to run the vendor dependent application. When the vendor dependent application exits it sends a VendorAppExited event to the main application to indicate that it has exited, the application can then use the VendorMode interface to communicate to other services that it is safe to regain controlf of the hardware.

The SetActiveInterface command can be used to communicate to the service which interface it should start on, this could be a local front screen, back screen or a remote screen on a terminal or mobile device. GetActiveInterface reports the currently active interface. Note that the interface can also be changed while the vendor dependent application is running.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

25.1 Command Messages

25.1.1 VendorApplication.StartLocalApplication

This command is issued by an application to start a local application which provides vendor dependent services. It can be used in conjunction with the Vendor Mode interface to manage vendor independent services and start vendor specific services, e.g. maintenance oriented applications.

Command Message

Payload	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"appName": Add example to YAML,`	string	
`"accessLevel": "accessBasic"`	string	
`}`		

Properties

timeout
Timeout in milliseconds for the command to complete. If set to zero, the command will not timeout but can be canceled.
default: 0

appName
Defines the vendor dependent application to start.

accessLevel
If specified, this defines the access level for the vendor dependent application interface. If not specified then the service will determine the level of access available. If the level of access is to be changed then an application exit should be performed, followed by a restart of the application specifying the new level of access. Specified as one of the following:

- `accessBasic` - The vendor dependent application is active for the basic access level. Once the application is active it will show the user interface for the basic access level.
- `accessIntermediate` - The vendor dependent application is active for the intermediate access level. Once the application is active it will show the user interface for the intermediate access level.
- `accessFull` - The vendor dependent application is active for the full access level. Once the application is active it will show the user interface for the full access level.

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	string	
`}`		

Properties

completionCode
The completion code.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Properties
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

None

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

25.1.2 VendorApplication.GetActiveInterface

This command is used to retrieve the interface that should be used by the vendor dependent application.

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 canceled.
default: 0

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"activeInterface": "consumerInterface"`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

activeInterface
Specifies the active interface as one of the following values:

- `consumerInterface` - The consumer interface.
- `operatorInterface` - The operator interface.

Event Messages

None

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

25.1.3 VendorApplication.SetActiveInterface

This command is used to indicate which interface should be displayed by a vendor dependent application. An application can issue this command to ensure that a vendor dependent application starts on the correct interface, or to change the interface while running.

Command Message

Payload	Type	Required
`{`		
`"activeInterface": "consumerInterface"`	string	
`}`		

Properties

activeInterface
Specifies the active interface as one of the following values:

- `consumerInterface` - The consumer interface.
- `operatorInterface` - The operator interface.

Completion Message

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML`	string	
`}`		

Properties

completionCode
The completion code.

errorDescription
If included, this contains additional vendor dependent information to assist with problem resolution.

Event Messages

- VendorApplication.InterfaceChangedEvent

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

25.2 Event Messages

25.2.1 VendorApplication.InterfaceChangedEvent

This event is used to indicate that the required interface has changed. This can be as a result of a SetActiveInterface command, or when the active interface is changed through vendor dependent means while the vendor dependent application is active. The activeInterface property indicates which interface has been selected.

Note: Applications must be prepared to allow the vendor dependent application to display on the active interface. This means that applications should no longer try to be the foreground or topmost window to ensure that the vendor dependent application is visible.

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"activeInterface": "consumerInterface"`	string	
`}`		

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
activeInterface Specifies the active interface as one of the following values: • `consumerInterface` - The consumer interface. • `operatorInterface` - The operator interface.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

25.3 Unsolicited Messages

25.3.1 VendorApplication.VendorAppExited

This event is used to indicate the vendor dependent application has exited, allowing an application the opportunity to exit Vendor Mode.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

25.3.2 VendorApplication.InterfaceChanged

This event is used to indicate that the required interface has changed. This can be as a result of a SetActiveInterface command, or when the active interface is changed through vendor dependent means while the vendor dependent application is active. The activeInterface property indicates which interface has been selected.

Note: Applications must be prepared to allow the vendor dependent application to display on the active interface. This means that applications should no longer try to be the foreground or topmost window to ensure that the vendor dependent application is visible.

Payload	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": Add example to YAML,`	string	
`"activeInterface": "consumerInterface"`	string	
`}`		

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
activeInterface Specifies the active interface as one of the following values: • `consumerInterface` - The consumer interface. • `operatorInterface` - The operator interface.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

26. 3.x Migration

This chapter provides information on how to migrate to and from XFS 3.x.

26.1 Summary

Migration to and from 3.x is a key requirement of XFS4IoT. This chapter describes how functionality available in XFS 3.x can be provided and/or derived to and from XFS4IoT.

26.2 General Information

26.2.1 CDM (Cash Dispense Module)

The CDM class provides access to cash dispensing functionality. In XFS4IoT, it has been restructured to use the CashDispenser, Storage and CashManagement interfaces.

WFS_INF_CDM_CASH_UNIT_INFO

Cash unit storage information is defined by the WFS_INF_CDM_CASH_UNIT_INFO category which is replaced in XFS4IoT by Storage.GetStorage. The following table lists the output fields in WFS_INF_CDM_CASH_UNIT_INFO and how they are mapped in XFS4IoT.

The concept of logical and physical cash units does not exist in XFS4IoT, only physical units are reported. The 3.x logical cash counts can be easily calculated from the given cash counts as described below.

Storage.SetStorage provides all the same functionality as the 3.x command but also provides additional functionality:

1. Only fields which need to be changed need to be provided, including only modifying storage units which are changing.
2. Storage.GetStorage provides a much more detailed breakdown of the capabilities and status of the storage units than the 3.x equivalents, therefore it is possible to decide in advance what a storage unit can be configured to do using *Storage.SetStorage*.
3. *Storage.SetStorage* only accepts as input the fields which can be modified.

3.x	**XFS4IoT**
WFSCDMCUINFO::usTellerID	Not supported, equivalent to 0 in XFS 3.x
WFSCDMCUINFO::usCount	Number of storage items with a cash interface
WFSCDMCASHUNIT::usNumber	index
WFSCDMCASHUNIT::usType	types. XFS4IoT has the option to support multiple types which is not available in 3.x, but an intelligent map can be performed, for example if the unit supports *cashIn* and *cashOut* then it can be mapped to WFS_CDM_TYPERECYCLING.
WFSCDMCASHUNIT::lpszCashUnitName	name
WFSCDMCASHUNIT::cUnitID	Storage unit property name
WFSCDMCASHUNIT::cCurrencyID	currency

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

3.x	XFS4IoT
WFSCDMCASHUNIT::ulValues	value. An additional conversion may be required due to the different types of these two items. *value* is the absolute value of the cash item, whereas *ulValues* is the value expressed in minimum dispense units. If the *value* of any of the items supported by the device can't be expressed as a C ULONG (integer between 0 and 0xffffffff) then a currency exponent will be required - see see CashManagement.GetBankNoteTypes.
WFSCDMCASHUNIT::ulInitialCount	Total number of items in initial
WFSCDMCASHUNIT::ulCount	If not a retract unit, count If a retract unit, retractOperations
WFSCDMCASHUNIT::ulRejectCount	Total number of items in rejected
WFSCDMCASHUNIT::ulMinimum	lowThreshold
WFSCDMCASHUNIT::ulMaximum	highThreshold
WFSCDMCASHUNIT::bAppLock	appLockOut
WFSCDMCASHUNIT::usStatus	status if not *ok*, otherwise replenishmentStatus
WFSCDMCASHUNIT::usNumPhysicalCUs	No support for logical units in XFS4IoT. This is equivalent to XFS 3.x value 1.
WFSCDMCASHUNIT::ulDispensedCount	Add the total number of items in all of the out fields.
WFSCDMCASHUNIT::ulPresentedCount	Total number of items in presented
WFSCDMCASHUNIT::ulRetractedCount	Total number of items in retracted
WFSCDMPHCU::lpPhysicalPositionName	positionName
WFSCDMPHCU::cUnitID	Storage unit property name
WFSCDMPHCU::ulInitialCount	Total number of items in initial
WFSCDMPHCU::ulCount	0 if a retract unit which cannot count items during a retract. In all other cases, start with count and subtract the total number of items in stacked, transport and unknown. If this results in a negative number, this should be 0.
WFSCDMPHCU::ulRejectCount	Total number of items in rejected
WFSCDMPHCU::ulMaximum	capacity
WFSCDMPHCU::usPStatus	status if not *ok*, otherwise replenishmentStatus Note this is a change from XFS 3.x where physical status is not overridden by count thresholds (ulMaximum and ulMinimum).
WFSCDMPHCU::bHardwareSensor	hardwareSensor
WFSCDMPHCU::ulDispensedCount	Add the total number of items in all of the out fields.
WFSCDMPHCU::ulPresentedCount	Total number of items in presented
WFSCDMPHCU::ulRetractedCount	Total number of items in retracted

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

26.2.2 CIM (Cash-In Module)

The CIM class provides access to cash accepting functionality. In XFS4IoT, it has been restructured to use the CashAcceptor, Storage and CashManagement interfaces.

WFS_INF_CIM_CASH_UNIT_INFO

Cash unit storage information is defined by the WFS_INF_CIM_CASH_UNIT_INFO category which is replaced in XFS4IoT by Storage.GetStorage. The following table lists the output fields in WFS_INF_CIM_CASH_UNIT_INFO and how they are mapped in XFS4IoT.

The concept of logical and physical cash units does not exist in XFS4IoT, only physical units are reported. The 3.x logical cash counts can be easily calculated from the given cash counts as described below.

Storage.SetStorage provides all the same functionality as the 3.x command but also provides additional functionality:

1. Only fields which need to be changed need to be provided, including only modifying storage units which are changing.
2. Storage.GetStorage provides a much more detailed breakdown of the capabilities and status of the storage units than the 3.x equivalents, therefore it is possible to decide in advance what a storage unit can be configured to do using *Storage.SetStorage*.
3. *Storage.SetStorage* only accepts as input the fields which can be modified.

3.x	**XFS4IoT**
WFSCIMCASHINFO::usCount	Number of storage items with a cash interface
WFSCIMCASHIN::usNumber	index
WFSCIMCASHIN::fwType	types. XFS4IoT has the option to support multiple types which is not available in 3.x, but an intelligent map can be performed, for example if the unit supports *cashIn* and *cashOut* then it can be mapped to WFS_CIM_TYPERECYCLING.
WFSCIMCASHIN::fwItemType	items
WFSCIMCASHIN::cUnitID	Storage unit property name
WFSCIMCASHIN::cCurrencyID	currency
WFSCIMCASHIN::ulValues	value. An additional conversion may be required due to the different types of these two items. *value* is the absolute value of the cash item, whereas *ulValues* is the value expressed in minimum dispense units. If the *value* of any of the items supported by the device can't be expressed as a C ULONG (integer between 0 and 0xffffffff) then a currency exponent will be required - see CashManagement.GetBankNoteTypes.
WFSCIMCASHIN::ulCashInCount	Add the total number of items in all of the in fields.
WFSCIMCASHIN::ulCount	If not a retract unit, count If a retract unit, retractOperations
WFSCIMCASHIN::ulMaximum	highThreshold
WFSCIMCASHIN::usStatus	status if not *ok*, otherwise replenishmentStatus
WFSCIMCASHIN::bAppLock	appLockIn

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

3.x	XFS4IoT
WFSCIMCASHIN::lpNoteNumberList	Combination of: 1. Start with initial 2. Add in 3. Remove out
WFSCIMCASHIN::usNumPhysicalCUs	No support for logical units in XFS4IoT. This is equivalent to XFS 3.x value 1.
WFSCIMCASHIN::lpszExtra	Any additional vendor-specific properties included in this storage unit not defined by the schema can be added to this field.
WFSCIMCASHIN::lpusNoteIDs	A list of all the noteID fields from *configuration*.
WFSCIMCASHIN::usCDMType	types. XFS4IoT has the option to support multiple types which is not available in 3.x, but an intelligent map can be performed, for example if the unit supports *cashIn* and *cashOut* then it can be mapped to WFS_CIM_TYPERECYCLING.
WFSCIMCASHIN::lpszCashUnitName	name
WFSCIMCASHIN::ulInitialCount	Total number of items in initial
WFSCIMCASHIN::ulDispensedCount	Add the total number of items in all of the out fields.
WFSCIMCASHIN::ulPresentedCount	Total number of items in presented
WFSCIMCASHIN::ulRetractedCount	Total number of items in retracted
WFSCIMCASHIN::ulRejectCount	Total number of items in rejected
WFSCIMCASHIN::ulMinimum	lowThreshold
WFSCIMPHCU::lpPhysicalPositionName	positionName
WFSCIMPHCU::cUnitID	Storage unit property name
WFSCIMPHCU::ulCashInCount	Add the total number of items in all of the in fields.
WFSCIMPHCU::ulCount	0 if a retract unit which cannot count items during a retract. In all other cases, start with count and subtract the total number of items in stacked, transport and unknown. If this results in a negative number, this should be 0.
WFSCIMPHCU::ulMaximum	capacity
WFSCIMPHCU::usPStatus	status if not *ok*, otherwise replenishmentStatus Note this is a change from XFS 3.x where physical status is not overridden by count thresholds (ulMaximum and ulMinimum).
WFSCIMPHCU::bHardwareSensors	hardwareSensor
WFSCIMPHCU::lpszExtra	Any additional vendor-specific properties included in this storage unit not defined by the schema can be added to this field.
WFSCIMPHCU::ulInitialCount	Total number of items in initial
WFSCIMPHCU::ulDispensedCount	Add the total number of items in all of the out fields.
WFSCIMPHCU::ulPresentedCount	Total number of items in presented
WFSCIMPHCU::ulRetractedCount	Total number of items in retracted
WFSCIMPHCU::ulRejectCount	Total number of items in rejected

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

WFS_SRVE_CIM_COUNTACCURACYCHANGED

The count accuracy is reported as part of the Storage for a cash unit, therefore when the count accuracy changes, a Storage.StorageChangedEvent will be generated.

© 2021 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.