CEN

WORKSHOP

AGREEMENT

CWA 17852

February 2022

ICS 35.240.40

English version

Extensions for Financial Services (XFS) - XFS4IoT Specification - Release 2021-1 Release Candidate

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

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

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

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

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

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

CEN-CENELEC Management Centre: Rue de la Science 23, B-1040 Brussels

Ref. No.:CWA 17852:2022 E

Table of Contents

Foreword

This CEN Workshop Agreement (CWA 17852:2022)has been developed in accordance with the CEN-CENELEC Guide 29 “CEN/CENELEC Workshop Agreements – A rapid prototyping to standardization” and with the relevant provisions of CEN/CENELEC Internal Regulations - Part 2. It was approved by a Workshop of representatives of interested parties on 2022-01-10, the constitution of which was supported by CEN following several public call for participation, the first of which was made on 1998-06-24. However, this CEN Workshop Agreement does not necessarily include all relevant stakeholders.

The final text of this CEN Workshop Agreement was provided to CEN for publication on 2021-01-12.

The following organizations and individuals developed and approved this CEN Workshop Agreement:

- ATM Japan LTD
- AURIGA SPA
- BANK OF AMERICA
- CASHWAY TECHNOLOGY
- CHINAL ECTRONIC FINANCIAL EQUIPMENT SYSTEM CO.
- CIMA SPA
- CLEAR2PAY SCOTLAND LIMITED
- DIEBOLD NIXDORF
- EASTERN COMMUNICATIONS CO. LTD – EASTCOM
- FINANZ INFORMATIK
- FUJITSU FRONTECH LIMITED
- FUJITSU TECHNOLOGY
- GLORY LTD
- GRG BANKING EQUIPMENT HK CO LTD
- HESS CASH SYSTEMS GMBH & CO. KG
- HITACHI OMRON TS CORP.
- HYOSUNG TNS INC
- JIANGSU GUOGUANG ELECTRONIC INFORMATION TECHNOLOGY
- KAL
- KEBA AG
- NCR FSG
- NEC CORPORATION
- OKI ELECTRIC INDUSTRY SHENZHEN
- OKI ELECTRONIC INDUSTRY CO

- PERTO S/A
- REINER GMBH & CO KG
- SALZBURGER BANKEN SOFTWARE
- SIGMA SPA
- TEB
- ZIJIN FULCRUM TECHNOLOGY CO

Attention is drawn to the possibility that some elements of this document may be subject to patent rights. CEN-CENELEC policy on patent rights is described in CEN-CENELEC Guide 8 “Guidelines for Implementation of the Common IPR Policy on Patent”. CEN shall not be held responsible for identifying any or all such patent rights.

Although the Workshop parties have made every effort to ensure the reliability and accuracy of technical and non-technical descriptions, the Workshop is not able to guarantee, explicitly or implicitly, the correctness of this document. Anyone who applies this CEN Workshop Agreement shall be aware that neither the Workshop, nor CEN, can be held liable for damages or losses of any kind whatsoever. The use of this CEN Workshop Agreement does not relieve users of their responsibility for their own actions, and they apply this document at their own risk. The CEN Workshop Agreement should not be construed as legal advice authoritatively endorsed by CEN/CENELEC.

1. Scope

XFS4IoT has been identified as a successor to XFS 3.x to meet the following requirements:

1. Replace the XFS and J/XFS standards in the marketplace.
2. Target industries – Retail Banking.
3. Operating System Agnostic and Technology and Language Adaptable.
4. Multi-Vendor – Able to run common core high level functionality on multiple vendors hardware, while providing access to finer level device API granularity.
5. Flexibility – enabling new hardware topologies, device types and functionality to be rapidly adapted.
6. Support end to end application level security.
7. Should not prevent the use of a low resource computing environment.
8. Provide a good developer experience by providing a well-documented API that is easy to learn, is quick to market and reduces risk by exposing an unambiguous interface.
9. Leverage existing standards.

Within the overall requirements specified in the Charter, the opportunity has been taken to solve some of the issues with the 3.x interface while retaining all the same functionality:

1. Binary data structures makes adding new functionality difficult due to compatibility issues, leading to multiple redundant versions of the same command appearing in many of the existing device classes. To resolve this, a flexible text based approach has been adopted including the wide use of default parameters.
2. Compound devices have been difficult for applications to implement, particularly cash recycling. Addition of other shared functionality such as end to end security would make the use of compound devices more prevalent. Compound devices are removed in XFS4IoT, a single Service can support as many interfaces as required to support its requirements.

Migration from and to 3.x is a major consideration to support adoption of XFS4IoT. While a lot of duplication has been removed (for example the Card Reader interface has fewer commands and events defined than the equivalent 3.x IDC specification), all the same IDC commands and events can be implemented. In some cases, this is achieved by having shared common commands such as Common.Status which replaces all the 3.x WFS_INF_XXX_STATUS commands.

2. API

This chapter defines the API functionality and messages. It defines the XFS4IoT API including but not limited to:

- System Architecture
- Message Definition
- End to End Security

XFS4IoT defines a system consisting of Services provided by one or more vendors. Each Service can support one or more interfaces as required to meet the requirements of the device or function it supports, so for example a Cash Recycling device will need the following interfaces to supply all the device's functionality:

- Common, which defines functionality common to all devices
- CashManagement, which defines functionality common to all cash handling devices
- CashAcceptor, which defines functionality common to all cash accepting devices
- CashDispenser, which defines functionality common to all cash dispensing devices
- Storage, which defines functionality common to devices which store items

Additional interfaces can be added as required for example KeyManagement to support encryption key management.

The following sections describe how clients and services create connections and send messages to each other.

2.1 References

ID	Description
api-1	JSON (https://www.json.org/)
api-2	XFS Interface Specification, End to End (E2E) for XFS/XFS4IoT Programmer's Reference
api-3	WebSockets - IETF RFC 6455

2.2 WebSockets Connections

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 (ref. api-3) 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 in the system, 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.

2.2.1 Overview

In this standard there are two types of "endpoint"; publisher and service. Each endpoint, of either type, is published by a single software/hardware vendor. A publisher endpoint is used for service discovery, to discover service endpoints. A single service endpoint 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 endpoints from different vendors. A "client" application may consume multiple services from multiple service endpoints 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 endpoint 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 endpoints. For each publisher endpoint it then exchanges JSON messages across WebSockets with URIs using a known format to recover a list of services published by that endpoint. 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 Uniform Resource Identifier (URI)

This section describes the Uniform Resource Identifiers used in XFS4IoT.

URI Format

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 URI consists of the following components:

URI Component	Description
wss:// or *ws://*	The protocol id for secure WebSockets. See Network Protocol.
machinename	The identification of the machine publishing endpoints. See Machine Identification.
portnumber	The port number discovered through the initial service discovery process - see Port Sequence.
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 protocol being used by this service. This may be updated to support services with different protocol versions in future versions of the specification and allows support for multiple versions of the specification on the same machine and endpoint. 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 and will have no impact on the protocol. This means that changes to the version field of the URI will be very rare. It will only be changed if there is a breaking, incompatible change or a fundamental change to the API. Because of this there won't be any need for complex version negotiation between the client and the service. The client will simply attempt to open the version of the API that it supports.
servicename	This will be included in the URI to allow different services to be identified on the same port. Services will normally match individual devices. The exact service name is discovered during service discovery and is vendor dependent. The format of the service name shouldn't be assumed. The only URI that doesn't include a service name is the service discovery URI.

For example, a service discovery URI might be:

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

Service URI might be:

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

The URI will be case sensitive and lower case.

Network Protocol

The WebSocket protocol defines two URI schemes, *wss* and *ws* that are used for encrypted and unencrypted connections. All connections in XFS4IoT should use the *wss* scheme using TLS encryption to secure network connections. The only exception will be when the network connection between the client and service can be physically secured, for example inside an ATM enclosure. In that case it will be possible to use clear

communication without TLS encryption and it is the responsibility of the hardware vendor to ensure that this is sufficient.

- Encrypted connections are identified by the **wss://** protocol specifier.
- Unencrypted connections are 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 specification 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 Certificate Authority or certificates issued by a vendor, which might limit access to the service.

A *wss* connection with invalid certificates will be invalid and will be rejected by both the client and the service.

Machine Identification

Machines publishing services are identified by URIs. Machines exposing endpoints 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.

Port Sequence

Services will be published on a sequence of IP ports consisting of port 80 or 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.

2.2.3 Service Publishing

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

Free Endpoint 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 hardware vendor 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. Therefore any single hardware vendor **must** publish all services on a single port, determined dynamically as above.

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

Handling Incoming Connections

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

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

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

The initial connection will be to the URI `wss://machinename:port/xfs4iot/v1.0`. This connection will then be used to list/discover individual services using the process outlined in Service Endpoint Discovery.

2.2.4 Service Discovery

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 endpoint discovery, then through service discovery for each service endpoint. It will do this through the following process.

Publisher Endpoint Discovery

The client will enumerate endpoints by attempting to open a WebSocket connection to the following URL on each port in the Port sequence.

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

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

The client will also start service endpoint 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 discovery, 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 client.

Service Endpoint Discovery

Once a connection has been established between the client and each publisher endpoint, the client will discover the services published by sending a service discovery command and receiving messages in the usual way as described in Message Types.

The only command sent to the publisher endpoint will be ServicePublisher.GetServices.

The publisher will acknowledge the command.

The command will be followed by zero or more ServicePublisher.ServiceDetailEvent messages, then complete with a completion message. Each event and the completion message will contain the following payload:

```
{
  "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 endpoint URI will be returned as a *serviceURI* property.

A publisher service may be designed to send one URI per message, or it may group URI together into a smaller number of messages. The publisher should try and send messages 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 messages. Alternatively, the URI may not be known straight away, for example 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 only 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 message 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.

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

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

Each service will endeavor to accept connections as quickly as possible during startup and restarts. Some devices are physically slow 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. Once a connection has been accepted a service may continue to report device as *starting* until the device is physically started and ready. While *starting*, any command on the connection other than Common.Status will fail with sequenceError.

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.

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.

2.3 Messages

XFS4IoT Services are accessed using messages passed over a WebSocket Interface. The messages are JSON formatted data Ref. api-1.

2.3.1 API Definition

All messages follow the same JSON structure consisting of two mandatory properties:

- **header** : containing properties that are common across all messages to allow the payload to be identified
- **payload** : containing properties that are specific to the message. It should be noted that not all messages require payload properties, so the payload may be empty.

These are the only two properties defined at the top level of the JSON structure as illustrated in the example below.

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

2.3.2 Header Definition

Headers are consistent across all XFS4IoT messages. All of the following properties are required.

Property	Property Type	Description
type	string	The message type. See Message Types.
name	string	The message name, for example *Common.Status*.
requestId	integer	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 0. For all other message types this will be non-zero. The client will supply values that are positive, incremental and greater than 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 *Common.Status* command message.

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

2.3.3 Payload Definition

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

2.3.4 Additional Properties

It is possible to include additional properties not defined by the specification. This can be useful in some cases and is allowed as long as those additional properties do not impact the proper functioning of the service or client.

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

When non-standard properties are used there's a risk that the same name could be used by different implementations, causing unexpected behaviors. Implementors should reduce the risk of this by using a company name or code as a prefix for the property. For example, a company called "Acme" might add properties 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.

Requirements

- Any additional property not defined by this specification and not recognized by the Service or the Client will be ignored.
- Ignoring an unknown property will have no effect on the standard behavior of the service or client. There will be no requirement to use undefined additional properties.
- The service or client may use undefined additional properties for whatever purpose they require. Dependance on undefined additional properties will mean the client or service is non-standard and may impact interoperability.
- Implementers should pick undefined additional property names to avoid name clashes.

2.4 Message Types

XFS4IoT supports the following message types.

type	**Direction**	**Description**
command	client to Service	Message sent to the Service to perform a command.
acknowledge	Service to client	Message from the Service indicating if the command is valid and queued.
event	Service to client	Intermediate message from the Service indicating progress of the command.
completion	Service to client	Message from the Service indicating the command is complete.
unsolicited	Service to client	Message from the Service unrelated to a command.

2.4.1 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 uses the standard header properties with *type* set to *command*.

The *requestId* is given by the client and allows the client to link messages sent in response to the command back to the original command. For example, the completion message for this command will contain the same *requestId*.

The *requestId* must be greater than or equal to 1 and incremented between each command, 0 is reserved for unsolicited events. The client is responsible for ensuring that each *requestId* is unique for a single connection. They do not have to be unique across connections. The request is identified by a combination of the *requestId* and the connection.

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

Examples of commands with payloads are shown in the example sequence.

2.4.2 Acknowledge Messages

As soon as the service has received and decoded the command message it will send an acknowledge message to indicate that the command message has been received and queued. This will normally include the *requestId* so that the client can identify which command it relates to (unless an error occurs which prevents the *requestId* being included). The message uses the standard header properties with *type* set to *acknowledge*.

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 does not.

Receiving the acknowledge message does not 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 message for the command, not in the acknowledge.

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

The acknowledge message contains the following standard payload properties.

Property	Property Type	Description
status	string	One of the status codes listed below. The status 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 *errorDescription* non-standard debugging properties in the acknowledge message.
errorDescription	string	If *status* is not *ok* this will give a human readable description of what caused the error. This may include details which help diagnose the cause. The format of this string should not be relied on.

status codes:

- `ok` - The command has been accepted for execution and will complete with a completion message.
- `invalidMessage` - Message cannot be decoded
- `invalidRequestID` - *requestId* is not greater than 0, or not greater than the previous *requestId*.
- `tooManyRequests` - There are currently too many requests queued by the service and the service cannot queue any more.

Examples of acknowledge messages are shown in the example sequence.

2.4.3 Event Messages

During the processing of the command the service can send multiple solicited events, as defined in the interface chapters. 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 *requestId* 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 does not recognize.

Examples of event messages are shown in the example sequence.

2.4.4 Completion Messages

If a command is accepted, there will be one completion 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 message will be sent.

The message uses the standard header properties with *type* set to *completion*. The completion message will contain the *requestId* from the original command message, so that the client can link the message back to the command. After the completion message for a command has been sent with a particular *requestId*, no more messages will be sent with that *requestId*.

Each completion message 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 message. When a command results in particular information, like reading a card, then that information is included in the completion message. The exact information included in each completion message is defined in the interface document that defines that completion message.

Examples of completion messages are shown in the example sequence.

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 chapter may 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 values defined in Completion Codes.

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

Completion Codes

After a command message has been received and associated acknowledge message 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 *requestId* 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 device's current state.
- `authorizationRequired` - The command request cannot be performed because it requires authorization.
- `noCommandNonce` - The value of the nonce stored in the hardware was cleared, for example by a power failure.
- `invalidToken` - The security token is invalid.
- `invalidTokenNonce` - The value of the nonce in the security token does not match the stored value.
- `invalidTokenHMAC` - The value of the HMAC in the security token is incorrect.
- `invalidTokenFormat` - The token format version value is not recognized, or the token format is somehow invalid.
- `invalidTokenKeyNoValue` - The key used for the HMAC for a token has not been loaded and the token cannot be validated.

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

2.4.5 Unsolicited Event Messages

The 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. The message uses the standard header properties with *type* set to *unsolicited*.

To allow clients to react to events quickly, unsolicited messages should be sent as soon as possible. For example, it should avoid queuing events until after the current command has been processed if it does not have to.

Since unsolicited events are not linked to command handling, they do not have a matching *requestId*. The event header will contain a *requestId* of 0. Unsolicited events are also broadcast to all clients, on all open connections.

Each interface chapter defines the unsolicited events relevant to the interface.

For compatibility with future specification changes, and to permit custom extensions by service implementors, the client should ignore any events that it does not recognize.

Examples of unsolicited messages are shown in the example sequence.

2.5 Command Processing

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

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

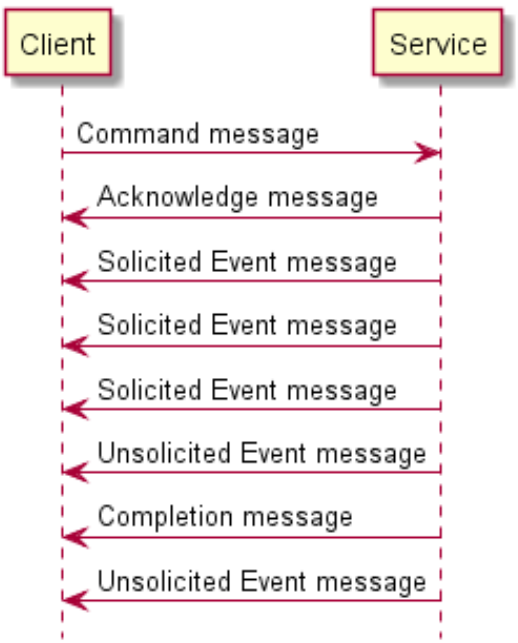
Services may also send unsolicited events directly to the client. This can happen at any time that the service connection is open. This could be during the processing of a command, or between commands.

The following sections provide information on various topics related to command processing:

- Standard command processing flow
- Command queuing
- Commands cancellation
- Example flow

2.5.1 Standard Sequence

The normal command message sequence will be as follows, note this example has multiple solicited and unsolicited events:



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

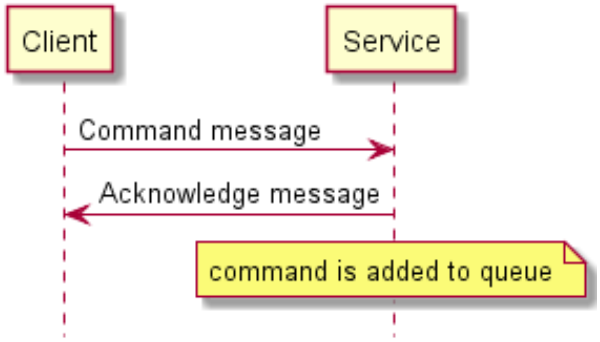
2.5.2 Command Queuing

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

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

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.CryptoData* 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*

2.5.3 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 *requestId* when sending the command message, and the service will send both acknowledge and completion

messages using that *requestId*. The Service will not send any event messages related to the *Common.Cancel* command *requestId*.

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

The *Common.Cancel* command itself cannot be canceled. Similarly, a *requestId* that does not match a queued or executing command *requestId* will have no effect.

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

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

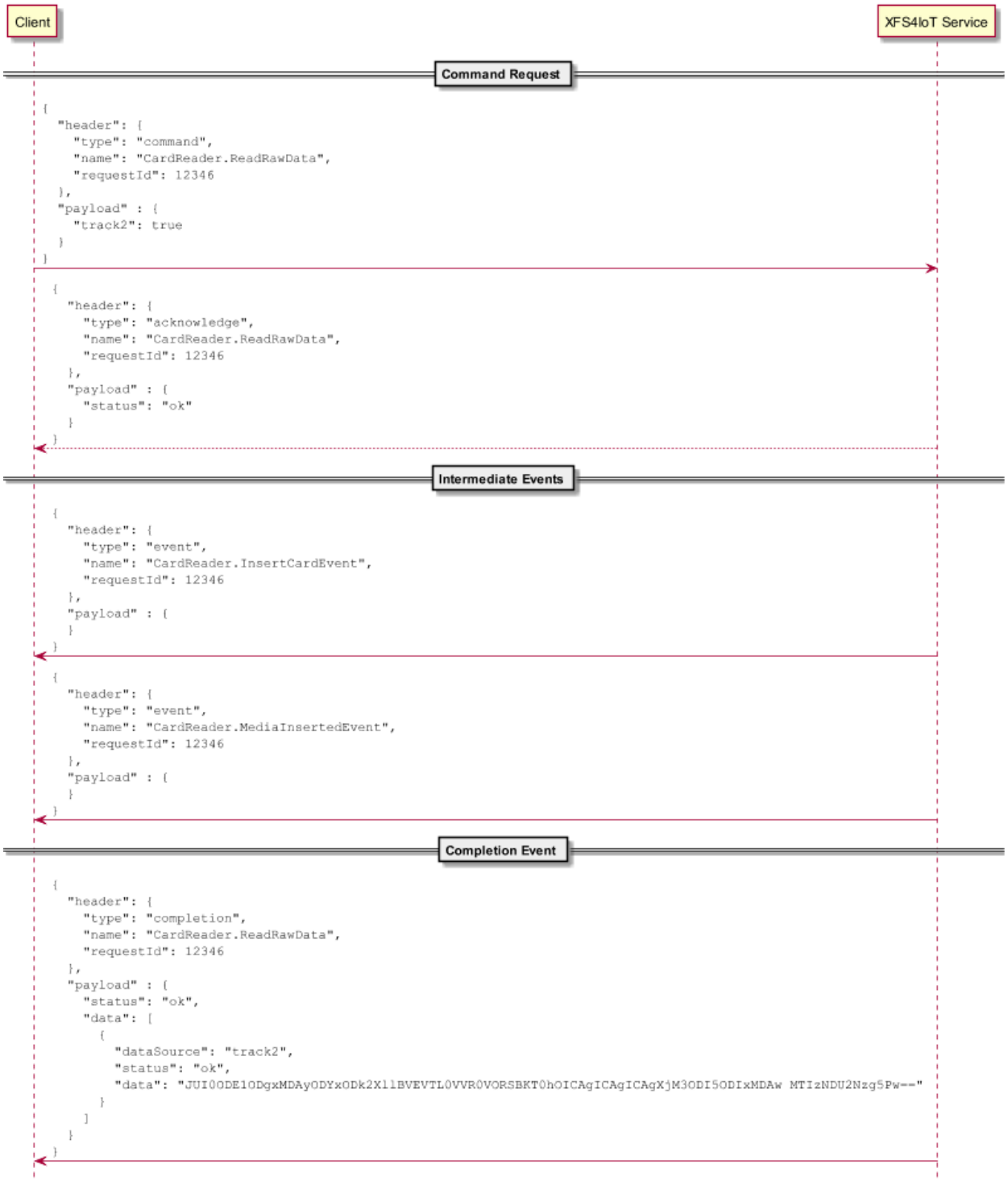
Clients should expect that, at some future point, commands may complete with a *completionCode* other than *canceled*. For example, device state prevents the command canceling forcing it to complete as if no cancel request had been received.

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

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

The Client should not attempt to cancel any one *requestId* 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.

2.5.4 Example Command Request Message Sequence



2.6 Message Versions

All messages types are assigned version numbers to enable evolution of individual messages. The major version number of a completion message will always be the same as the major version number of the command message it is associated with, however, the minor version numbers can be different.

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 to use the command message property and the value it should be set to.

Each release of the specification defines the message version numbers of the command, acknowledge, 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.

2.6.1 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 and minor version numbers are incremented according to the scope of change described in the following sections.

The major version must be greater than 0. If a minor change is made, the minor version is incremented and the major version remains the same. If a major change is made, the major version is incremented and the minor version is reset 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.

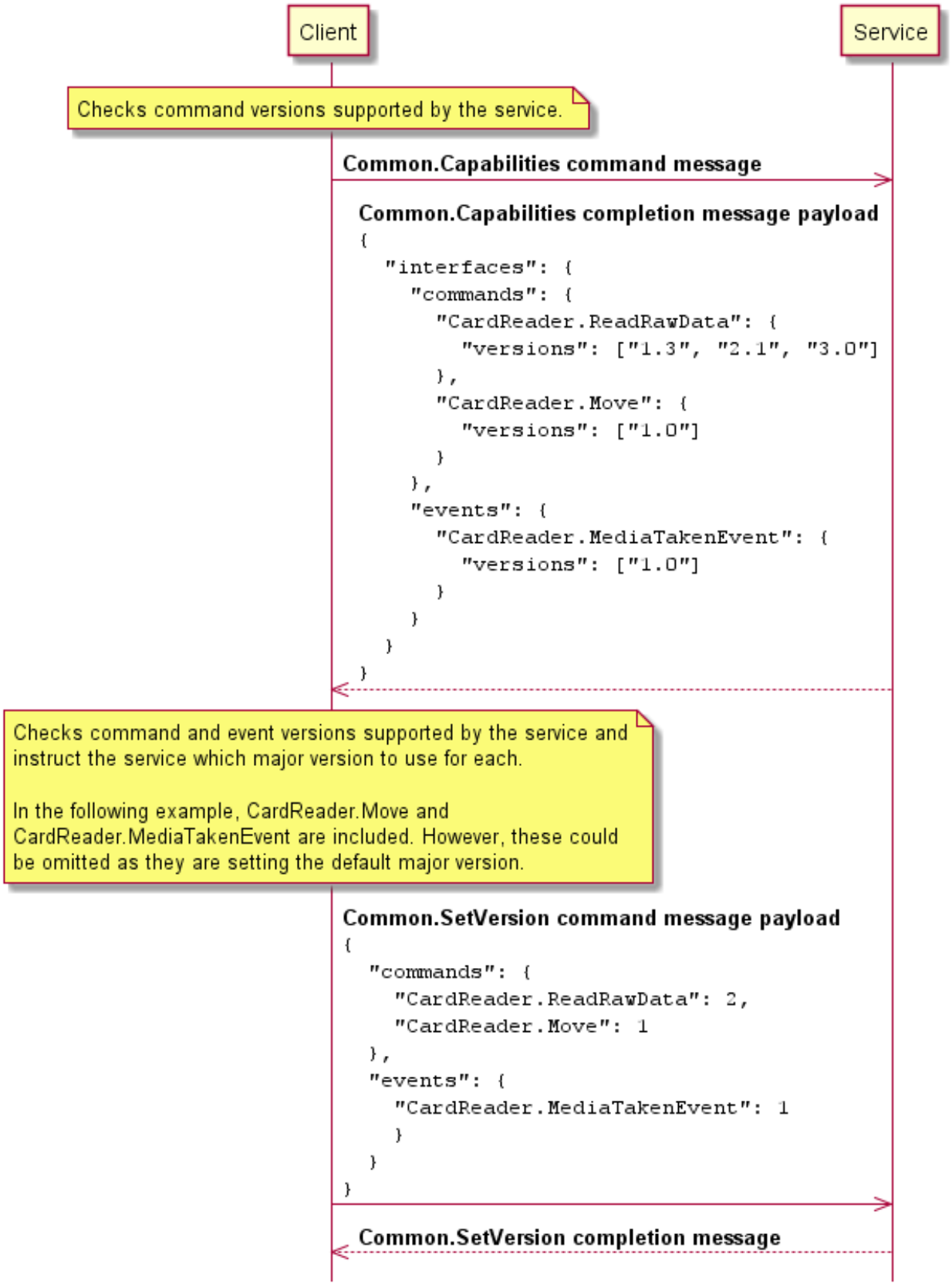
For additional completion, event and unsolicited message properties, clients should expect that new properties may be added and if not required, ignored. That is, clients should not break because they do not recognize additional properties.

2.6.2 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 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.



2.6.3 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	1.0 propA	1.0 propA
Minor update - event property added	1.1 propA propB	**1.1** propA propB	1.0 propA	1.0 propA
Major update - completion property removed - command unchanged - unsolicited property removed - unsolicited property added	**2.0** propA propB	1.1 propA propB	**2.0** propB	**2.0** propB

2.6.4 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"
}
```

2.7 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 authorized by an authorizing 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 authorizing host and the dispenser hardware.

Details of end-to-end (E2E) security are covered in the generic E2E security specification [Ref. api-2] shared between XFS3.x and XFS4IoT. Generic and specific E2E tokens are defined in that specification. 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 specification, including Common.GetCommandNonce and Common.ClearCommandNonce

3. Service Publisher Interface

This chapter defines the Service Publisher interface functionality and messages.

3.1 Command Messages

3.1.1 ServicePublisher.GetServices

Command sent to the service discovery port to identify services exposed by this publisher.

This command can be used while in Vendor Mode.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"vendorName": "ACME ATM Hardware GmbH",	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
Array of one or more services exposed by the publisher.

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

```
format: URI
```

Event Messages

- ServicePublisher.ServiceDetailEvent

3.2 Event Messages

3.2.1 ServicePublisher.ServiceDetailEvent

Details of one or more services published by this endpoint.

Event Message

Payload (version 1.0)	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
Array of one or more services exposed by the publisher.

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

```
format: URI
```

4. Common Interface

This chapter defines the Common interface functionality and messages.

4.1 Command Messages

4.1.1 Common.Status

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

This command can be used while in Vendor Mode.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		
Properties		
timeout Timeout in milliseconds for the command to complete. If set to 0, the command will not timeout but can be canceled. default: 0		

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"common": {	object	Yes
"device": "online",	string	Yes
"devicePosition": "inPosition",	string	
"powerSaveRecoveryTime": 0,	integer	
"antiFraudModule": "ok",	string	
"exchange": "notSupported",	string	
"endToEndSecurity": "enforced"	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	

Payload (version 1.0)	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	
}]		
},		
"cashDispenser": {	object	
"intermediateStacker": "empty",	string	
"positions": [{	array (object)	
"position": "outDefault",	string	
"shutter": "closed",	string	
"positionStatus": "empty",	string	
"transport": "ok",	string	
"transportStatus": "empty"	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": {	object	Yes
"activeAvailable": false,	boolean	
"inactiveAvailable": false	boolean	
}		
},		
"textTerminal": {	object	
"keyboard": "on",	string	Yes
"keyLock": "on",	string	Yes
"displaySizeX": 0,	integer	Yes
"displaySizeY": 0	integer	Yes

Payload (version 1.0)	Type	Required
},		
"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	
"*vendorSpecificPaperSupply*": "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	
},		
"barcodeReader": {	object	
"scanner": "on"	string	
},		
"biometric": {	object	
"subject": "present",	string	Yes
"capture": false,	boolean	Yes
"dataPersistence": "persist",	string	Yes
"remainingStorage": 0	integer	Yes
},		

Payload (version 1.0)	Type	Required
"camera": {	object	
"media": {	object	
"room": "ok",	string	
"person": "ok",	string	
"exitSlot": "ok",	string	
"exampleProperty1": "ok",	string	
"exampleProperty2": "ok"	string	
},		
"cameras": {	object	
"room": "notSupported",	string	
"person": "notSupported",	string	
"exitSlot": "notSupported",	string	
"exampleProperty1": "notSupported",	string	
"exampleProperty2": "notSupported"	string	
},		
"pictures": {	object	
"room": 0,	integer	
"person": 0,	integer	
"exitSlot": 0,	integer	
"exampleProperty1": 0,	integer	
"exampleProperty2": 0	integer	
}		
},		
"lights": {	object	
"cardReader": {	object	
"position": "left",	string	
"flashRate": "off",	string	
"color": "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.		

Payload (version 1.0)	Type	Required
},		
"passbookPrinter": {	object	
See cardReader properties.		
},		
"envelopeDepository": {	object	
See cardReader properties.		
},		
"billAcceptor": {	object	
See cardReader properties.		
},		
"envelopeDispenser": {	object	
See cardReader properties.		
},		
"documentPrinter": {	object	
See cardReader properties.		
},		
"coinAcceptor": {	object	
See cardReader properties.		
},		
"scanner": {	object	
See cardReader properties.		
},		
"contactless": {	object	
See cardReader properties.		
},		
"cardReader2": {	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	

Payload (version 1.0)	Type	Required
See cardReader properties.		
},		
"statusSupervisor": {	object	
See cardReader properties.		
},		
"statusInService": {	object	
See cardReader properties.		
},		
"fasciaLight": {	object	
See cardReader properties.		
},		
"*vendorSpecificLight*": {	object	
See cardReader properties.		
}		
},		
"auxiliaries": {	object	
"operatorSwitch": "notAvailable",	string	
"tamperSensor": "notAvailable",	string	
"internalTamperSensor": "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	
"rate": "on",	string	
"signal": "keypress"	string	
},		

Payload (version 1.0)	Type	Required
"heating": "notAvailable",	string	
"consumerDisplayBacklight": "notAvailable",	string	
"signageDisplay": "notAvailable",	string	
"volume": {	object	
"volumeLevel": 1	integer	
},		
"UPS": {	object	
"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	
}		
}		

Properties

completionCode
The completion code.

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

common
Status information common to all XFS4IoT services.

Properties
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 or breaking an interlock). • `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. • `starting` - The device is starting and performing whatever initialization is necessary. This can be reported after the connection is made but before the device is ready to accept commands. This must only be a temporary state, the Service must report a different state as soon as possible. If an error causes initialization to fail then the state should change to *hardwareError*.
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 0 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 if available. Following values are possible: • `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.StatusChangedEvent 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"

Properties
common/endToEndSecurity Specifies the status of end to end security support on this device. Also see Common.CapabilityProperties.endToEndSecurity. • `notSupported` - E2E security is not supported by this hardware. Any command can be called without a token. • `notEnforced` - E2E security is supported by this hardware but it is not currently enforced, for example because required keys aren't loaded. It's currently possible to perform E2E commands without a token. • `notConfigured` - E2E security is supported but not correctly configured, for example because required keys aren't loaded. Any attempt to perform any command protected by E2E security will fail. • `enforced` - E2E security is supported and correctly configured. E2E security will be enforced. Calling E2E protected commands will only be possible if a valid token is given. default: "notSupported"
cardReader Status information for XFS4IoT services implementing the CardReader interface. This will be omitted if the CardReader interface is not supported.
cardReader/media Specifies the 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. • `open` - 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 a 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.

Properties
cardReader/chipModule Specifies the state of the chip card module reader as one of the following: • `ok` - The chip card module is in a good state. • `inoperable` - The chip card module is inoperable. • `unknown` - The state of the chip card module cannot be determined. • `notSupported` - Reporting the chip card module status is not supported.
cardReader/magWriteModule Specifies the state of the magnetic card writer as one of the following: • `ok` - The magnetic card writing module is in a good state. • `inoperable` - The magnetic card writing module is inoperable. • `unknown` - The state of the magnetic card writing module cannot be determined. • `notSupported` - Reporting the magnetic card writing module status is not supported.
cardReader/frontImageModule Specifies the state of the front image reader as one of the following: • `ok` - The front image reading module is in a good state. • `inoperable` - The front image reading module is inoperable. • `unknown` - The state of the front image reading module cannot be determined. • `notSupported` - Reporting the front image reading module status is not supported.
cardReader/backImageModule Specifies the state of the back image reader as one of the following: • `ok` - The back image reading module is in a good state. • `inoperable` - The back image reading module is inoperable. • `unknown` - The state of the back image reading module cannot be determined. • `notSupported` - Reporting the back image reading module status is not supported.
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. The 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.CashInStart 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. The 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.

Properties
cashAcceptor/banknoteReader Supplies the state of the banknote reader. The 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 Cash Acceptor 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.
cashAcceptor/positions Array of structures reporting status for each position from which items can be accepted.
cashAcceptor/positions/position Supplies the input or output position as one of the following values. If not specified, the default position applies. Supported positions are reported in Common.Capabilities. • `inDefault` - Default input position. • `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. • `outDefault` - Default output 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. The following values are possible: • `closed` - The shutter is operational and is fully closed. • `open` - The shutter is operational and is open. • `jammedOpen` - The shutter is jammed, but fully open. It is not operational. • `jammedPartiallyOpen` - The shutter is jammed, but partially open. It is not operational. • `jammedClosed` - The shutter is jammed, but fully closed. It is not operational. • `jammedUnknown` - The shutter is jammed, but its position is unknown. It is not operational. • `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. The following values are possible: • `empty` - The position is empty. • `notEmpty` - The position is not empty. • `unknown` - Due to a hardware error or other condition, the state of the position cannot be determined. • `notSupported` - The device is not capable of reporting whether items are at the position.

Properties
cashAcceptor/positions/transport Supplies the state of the transport mechanism. The transport is defined as any area leading to or from the position. The following values are possible: • `ok` - The transport is in a good state. • `inoperative` - The transport is inoperative due to a hardware failure or media jam. • `unknown` - Due to a hardware error or other condition the state of the transport cannot be determined. • `notSupported` - The physical device has no transport or transport state reporting is not supported.
cashAcceptor/positions/transportStatus Returns information regarding items which may be on the transport. If the device is a recycler device it is possible that the transport will not be empty due to a previous dispense operation. The 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. • `unknown` - Due to a hardware error or other condition it is not known whether there are items on the transport. • `notSupported` - The device is not capable of reporting whether items are on the transport.
cashDispenser Status information for XFS4IoT services implementing the CashDispenser interface. This will be omitted if the CashDispenser interface is not supported.
cashDispenser/intermediateStacker Supplies the state of the intermediate stacker. These bills are typically present on the intermediate stacker as a result of a retract operation or because a dispense has been performed without a subsequent present. Following values are possible: • `empty` - The intermediate stacker is empty. • `notEmpty` - The intermediate stacker is not empty. The items have not been in customer access. • `notEmptyCustomer` - The intermediate stacker is not empty. The items have been in customer access. If the device is a recycler then the items on the intermediate stacker may be there as a result of a previous cash-in operation. • `notEmptyUnknown` - The intermediate stacker is not empty. It is not known if the items have been in customer access. • `unknown` - Due to a hardware error or other condition, the state of the intermediate stacker cannot be determined. • `notSupported` - The physical device has no intermediate stacker.
cashDispenser/positions Array of structures for each position to which items can be dispensed or presented.
cashDispenser/positions/position Supplies the output position as one of the following values. Supported positions are reported in Common.Capabilities. • `outDefault` - Default output 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. default: "outDefault"

Properties
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. • `jammedOpen` - The shutter is jammed, but fully open. It is not operational. • `jammedPartiallyOpen` - The shutter is jammed, but partially open. It is not operational. • `jammedClosed` - The shutter is jammed, but fully closed. It is not operational. • `jammedUnknown` - The shutter is jammed, but its position is unknown. It is not operational. • `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 position is empty. • `notEmpty` - The position is not empty. • `unknown` - Due to a hardware error or other condition, the state of the position cannot be determined. • `notSupported` - The device is not capable of reporting whether items are at the 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. • `unknown` - 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.
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.

Properties
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 empty, missing, inoperative or in a 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 storage unit is present, or when an application lock is set on every storage unit, or when counterfeit or suspect items 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.
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 inactive key beeping is reported independently.
keyboard/autoBeepMode/activeAvailable Specifies whether an automatic tone will be generated for all active keys. default: false
keyboard/autoBeepMode/inactiveAvailable Specifies whether an automatic tone will be generated for all inactive keys. default: false
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.

Properties
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 (the number of columns that can be displayed). Property value constraints: `minimum: 0`
textTerminal/displaySizeY 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`
printer Status information for XFS4IoT services implementing the Printer interface. This will be omitted if the Printer interface is not supported.
printer/media Specifies the state of the print media (i.e. receipt, statement, passbook, etc.) as one of the following values. This field does not apply to journal printers: • `notSupported` - The capability to report the state of the print media is not supported by the device. • `unknown` - The state of the print media cannot be determined with the device in its current state. • `present` - Media is in the print position, on the stacker or on the transport (i.e. a passbook in the parking station is not considered to be present). On devices with continuous paper supplies, this value is set when paper is under the print head. On devices with no supply or individual sheet supplies, this value is set when paper/media is successfully inserted/loaded. • `notPresent` - Media is not in the print position or on the stacker. • `jammed` - Media is jammed in the device. • `entering` - Media is at the entry/exit slot of the device. • `retracted` - Media was retracted during the last command which controlled media.
printer/paper Specifies the state of paper supplies as one of the following values. Omitted if not applicable: • `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.

Properties
printer/paper/vendorSpecificPaperSupply (example name)
printer/toner Specifies the state of the toner or ink supply or the state of the ribbon as one of the following: • `notSupported` - Capability not supported by device. • `unknown` - Status of toner or ink supply or the ribbon cannot be determined with device in its current state. • `full` - The toner or ink supply is full or the ribbon is OK. • `low` - The toner or ink supply is low or the print contrast with a ribbon is weak. • `out` - The toner or ink supply is empty or the print contrast with a ribbon is not sufficient any more.
printer/ink Specifies the status of the stamping ink in the printer as one of the following values: • `notSupported` - Capability not supported by device. • `unknown` - Status of the stamping ink supply cannot be determined with device in its current state. • `full` - Ink supply in device is full. • `low` - Ink supply in device is low. • `out` - Ink supply in device is empty.
printer/lamp Specifies the status of the printer imaging lamp as one of the following values: • `notSupported` - Capability not supported by device. • `unknown` - Status of the imaging lamp cannot be determined with device in its current state. • `ok` - The lamp is OK. • `fading` - The lamp should be changed. • `inop` - The lamp is inoperative.
printer/retractBins An array of bin state objects. If no retain bins are supported, the array will be empty.
printer/retractBins/state Specifies the state of the printer retract bin as one of the following: • `ok` - The retract bin of the printer is in a healthy state. • `full` - The retract bin of the printer is full. • `unknown` - Status cannot be determined with device in its current state. • `high` - The retract bin of the printer is nearly full. • `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 0 by the Printer.ResetCount command. Property value constraints: `minimum: 0`
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.

Properties
printer/paperType/external The external paper supply paper type.
printer/paperType/aux The auxililliary paper supply paper type.
printer/paperType/aux2 The second auxililliary paper supply paper type.
printer/paperType/park The parking station paper supply paper type.
printer/paperType/exampleProperty1 (example name)
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.
barcodeReader Status information for XFS4IoT services implementing the Barcode Reader interface. This will be omitted if the Barcode Reader 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. • `unknown` - 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). • `notSupported` - The physical device does not support the ability to report whether a subject is on the scanning position.
biometric/capture Indicates whether or not scanned biometric data has been captured using the Biometric.Readand 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.
biometric/dataPersistence Specifies the current data persistence mode. The data persistence mode controls how biometric data that has been captured using the Biometric.Read will be handled. The following values are possible: • `persist` - Biometric data captured using the Biometric.Read can persist until all sessions are closed, the device is power failed or rebooted, or the Biometric.Readis requested again. This captured biometric data can also be explicitly cleared using the Biometric.Clear or Biometric.Reset. • `clear` - Captured biometric data will not persist. Once the data has been either returned in the Biometric.Reador used by the Biometric.Match, then the data is cleared from the device.

Properties
biometric/remainingStorage Specifies how much of the reserved storage specified by the *templateStorage* capability is remaining for the storage of templates in bytes. if omitted, this property is not supported. Property value constraints: `minimum: 0`
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, this will be *notSupported*. • `ok` - The media is in a good state. • `high` - The media is almost full (threshold). • `full` - The media is full. • `notSupported` - 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/media/exampleProperty1 (example name)
camera/cameras Specifies the state of the cameras as one of the following. • `notSupported` - The camera is not supported. • `ok` - The camera is in a good state. • `inoperative` - 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.
camera/cameras/exampleProperty1 (example name)
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 property should be 0.
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`

Properties
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 Reader Light.
lights/cardReader/position The light position. Can be used for devices which have multiple input and output positions, omitted if not required. 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).
lights/cardReader/color The light color as one of the following values: • `red` - The light is red. • `green` - The light is green. • `yellow` - The light is 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.

Properties
lights/receiptPrinter Receipt Printer Light.
lights/passbookPrinter Passbook Printer Light.
lights/envelopeDepository Envelope Depository 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 Reader Light.
lights/cardReader2 Card Reader 2 Light.
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/vendorSpecificLight (example name) Additional vendor specific lights
auxiliaries Status information for XFS4IoT services implementing the Auxiliaries interface. This will be omitted if the Auxiliaries interface is not supported.

Properties
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` - The switch is in supervisor mode.
auxiliaries/tamperSensor Specifies the state of the Tamper sensor. • `notAvailable` - The status is not available. • `off` - There is no indication of a tampering attempt. • `on` - There has been a tampering attempt.
auxiliaries/internalTamperSensor 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 status 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 is not available. • `off` - The seismic activity has not been high enough to trigger the sensor. • `on` - The seismic or other activity has triggered the sensor.
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.

Properties
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.
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.

Properties
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 fully 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. • `partiallyOpen` - The Vandal Shield is partially open. • `jammed` - The Vandal Shield is jammed. • `tampered` - The Vandal Shield has potentially been tampered with.
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 rear 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.

Properties
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.
auxiliaries/audio Specifies the state of the Audio Indicator.
auxiliaries/audio/rate Specifies the state 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.
auxiliaries/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.
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.
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. 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 Specifies the state of the volume control. Omitted if not available.

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`

Properties
auxiliaries/UPS Specifies the state of the Uninterruptible Power Supply. Omitted if the status is not available. • `low` - The charge level of the UPS is low. • `engaged` - The UPS is engaged. • `powering` - The UPS is powering the system. • `recovered` - The UPS was engaged when the main power went off.
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.
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` - The 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.

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 whether the Microphone 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.

vendorMode/device

Specifies the status of the Vendor Mode Service. 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.

Properties
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

4.1.2 Common.Capabilities

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

This command can be used while in Vendor Mode.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"interfaces": [{	array (object)	Yes
"name": "Common",	string	
"commands": {	object	
"*CardReader.ReadRawData*": {	object	
"versions": ["1.3", "2.1", "3.0"]	array (string)	
},		
"*CardReader.Move*": {	object	
See CardReader.ReadRawData properties.		
}		
},		
"events": {	object	
"*CardReader.MediaInsertedEvent*": {	object	
"versions": ["1.3", "2.1", "3.0"]	array (string)	
},		
"*CardReader.MediaRemovedEvent*": {	object	
See CardReader.MediaInsertedEvent properties.		
}		
},		
"maximumRequests": 0	integer	
}],		
"common": {	object	Yes
"serviceVersion": "1.3.42",	string	

Payload (version 1.0)	Type	Required
"deviceInformation": [{	array (object)	
"modelName": "AcmeModel42",	string	
"serialNumber": "1.0.12.05",	string	
"revisionNumber": "1.2.3",	string	
"modelDescription": "Acme Dispenser Model 3",	string	
"firmware": [{	array (object)	
"firmwareName": "Acme Firmware",	string	
"firmwareVersion": "1.0.1.2",	string	
"hardwareRevision": "4.3.0.5"	string	
}],		
"software": [{	array (object)	
"softwareName": "Acme Software Name",	string	
"softwareVersion": "1.3.0.2"	string	
}]		
}],		
"powerSaveControl": false,	boolean	
"antiFraudModule": false,	boolean	
"endToEndSecurity": {	object	
"required": "always",	string	Yes
"hardwareSecurityElement": true,	boolean	Yes
"responseSecurityEnabled": "always",	string	Yes
"commands": ["CashDispenser.Dispense"],	array (string)	Yes
"commandNonceTimeout": 3600	integer	Yes
}		
},		
"cardReader": {	object	
"type": "motor",	string	
"readTracks": {	object	
"track1": false,	boolean	
"track2": false,	boolean	
"track3": false,	boolean	
"watermark": false,	boolean	
"frontTrack1": false,	boolean	
"frontImage": false,	boolean	
"backImage": false,	boolean	
"track1JIS": false,	boolean	
"track3JIS": false,	boolean	
"ddi": false	boolean	
},		
"writeTracks": {	object	
"track1": false,	boolean	

Payload (version 1.0)	Type	Required
"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	
"loco": false,	boolean	
"hico": false,	boolean	
"auto": false	boolean	
},		
"chipPower": {	object	
"cold": false,	boolean	
"warm": false,	boolean	
"off": false	boolean	
},		
"memoryChipProtocols": {	object	
"siemens4442": false,	boolean	
"gpm896": false	boolean	
},		
"positions": {	object	
"exit": false,	boolean	
"transport": false	boolean	
},		
"cardTakenSensor": false	boolean	
},		
"cashAcceptor": {	object	

Payload (version 1.0)	Type	Required
"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	
},		
"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	

Payload (version 1.0)	Type	Required
"reject": false,	boolean	
"billCassette": false,	boolean	
"cashIn": false	boolean	
},		
"cashInLimit": {	object	
"byTotalItems": false,	boolean	
"byAmount": false	boolean	
},		
"countActions": {	object	
"individual": false,	boolean	
"all": false	boolean	
},		
"counterfeitAction": "none"	string	
},		
"cashDispenser": {	object	
"type": "tellerBill",	string	
"maxDispenseItems": 1,	integer	
"shutterControl": false,	boolean	
"retractAreas": {	object	
"retract": false,	boolean	
"transport": false,	boolean	
"stacker": false,	boolean	
"reject": false,	boolean	
"itemCassette": false,	boolean	
"cashIn": false	boolean	
},		
"retractTransportActions": {	object	
"present": false,	boolean	
"retract": false,	boolean	
"reject": false,	boolean	
"itemCassette": false,	boolean	
"cashIn": false	boolean	
},		
"retractStackerActions": {	object	
"present": false,	boolean	
"retract": false,	boolean	
"reject": false,	boolean	
"itemCassette": false,	boolean	
"cashIn": false	boolean	
},		
"intermediateStacker": false,	boolean	

Payload (version 1.0)	Type	Required
"itemsTakenSensor": false,	boolean	
"positions": {	object	
"left": false,	boolean	
"right": false,	boolean	
"center": false,	boolean	
"top": false,	boolean	
"bottom": false,	boolean	
"front": false,	boolean	
"rear": false	boolean	
},		
"moveItems": {	object	
"fromCashUnit": false,	boolean	
"toCashUnit": false,	boolean	
"toTransport": false,	boolean	
"toStacker": false	boolean	
}		
},		
"cashManagement": {	object	
"cashBox": false,	boolean	
"exchangeType": {	object	
"byHand": false	boolean	
},		
"itemInfoTypes": {	object	
"serialNumber": false,	boolean	
"signature": false,	boolean	
"image": false	boolean	
},		
"classificationList": 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	

Payload (version 1.0)	Type	Required
"banksys": false,	boolean	
"emv": false,	boolean	
"iso3": false,	boolean	
"ap": false,	boolean	
"iso4": false	boolean	
},		
"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.		
}		

Payload (version 1.0)	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	
"M0": {	object	
"T": {	object	
"G": {	object	
"cryptoMethod": {	object	
"rsassaPkcs1V15": false,	boolean	

Payload (version 1.0)	Type	Required
"rsassaPss": false	boolean	
},		
"hashAlgorithm": {	object	
"sha1": false,	boolean	
"sha256": false	boolean	
}		
},		
"S": {	object	
See G properties.		
}		
},		
"R": {	object	
See T properties.		
}		
},		
"S0": {	object	
See M0 properties.		
}		
},		
"verifyAttributes": {	object	
"M0": {	object	
"T": {	object	
"V": {	object	
"cryptoMethod": {	object	
"rsassaPkcs1V15": false,	boolean	
"rsassaPss": false	boolean	
},		
"hashAlgorithm": {	object	
"sha1": false,	boolean	
"sha256": false	boolean	
}		
}		
},		
"R": {	object	
See T properties.		
}		
},		
"S0": {	object	
See M0 properties.		
}		
}		

Payload (version 1.0)	Type	Required
},		
"keyManagement": {	object	
"keyNum": 0,	integer	Yes
"derivationAlgorithms": {	object	
"chipZka": false	boolean	
},		
"keyCheckModes": {	object	Yes
"self": false,	boolean	
"zero": false	boolean	
},		
"hsmVendor": "Add example",	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
"randomNumber": false,	boolean	
"exportDeviceId": 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	

Payload (version 1.0)	Type	Required
},		
"keyBlockImportFormats": {	object	Yes
"A": false,	boolean	
"B": false,	boolean	
"C": false,	boolean	
"D": 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
"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	
"restrictedKeyUsage": "Add example"	string	

Payload (version 1.0)	Type	Required
},		
"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	
"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	

Payload (version 1.0)	Type	Required
},		
"hashAlgorithm": {	object	Yes
"sha1": false,	boolean	
"sha256": false	boolean	
}		
},		
"S": {	object	
See V properties.		
}		
},		
"R": {	object	
See T properties.		
}		
},		
"S0": {	object	
See M0 properties.		
}		
}		
},		
"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

Payload (version 1.0)	Type	Required
"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	
"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	

Payload (version 1.0)	Type	Required
"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	
"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": 5,	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	
"*exampleProperty1*": false,	boolean	
"*exampleProperty2": false*	boolean	
},		
"mediaTaken": false,	boolean	
"retractBins": 1,	integer	
"maxRetract": [0],	array (integer)	
"imageType": {	object	
"tif": false,	boolean	

Payload (version 1.0)	Type	Required
"wmf": false,	boolean	
"bmp": false,	boolean	
"jpg": false	boolean	
},		
"frontImageColorFormat": {	object	
"binary": false,	boolean	
"grayscale": false,	boolean	
"full": false	boolean	
},		
"backImageColorFormat": {	object	
"binary": false,	boolean	
"grayScale": false,	boolean	
"full": false	boolean	
},		
"codelineFormat": {	object	
"cmc7": false,	boolean	
"e13b": false,	boolean	
"ocr": false	boolean	
},		
"imageSource": {	object	
"imageFront": false,	boolean	
"imageBack": false,	boolean	
"codeLine": false	boolean	
},		
"dispensePaper": false,	boolean	
"osPrinter": "Add example",	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": "notSupported"	string	

Payload (version 1.0)	Type	Required
},		
"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	
"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	

Payload (version 1.0)	Type	Required
"rssExpanded": false,	boolean	
"rssRestricted": false,	boolean	
"compositeCodeA": false,	boolean	
"compositeCodeB": false,	boolean	
"compositeCodeC": false,	boolean	
"posiCodeA": false,	boolean	
"posiCodeB": false,	boolean	
"triopticCode39": false,	boolean	
"codablockF": false,	boolean	
"code16K": false,	boolean	
"qrCode": false,	boolean	
"aztec": false,	boolean	
"ukPost": false,	boolean	
"planet": false,	boolean	
"postnet": false,	boolean	
"canadianPost": false,	boolean	
"netherlandsPost": false,	boolean	
"australianPost": false,	boolean	
"japanesePost": false,	boolean	
"chinesePost": false,	boolean	
"koreanPost": false	boolean	
}		
},		
"biometric": {	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": 0,	integer	Yes
"dataFormats": {	object	Yes
"isoFid": false,	boolean	

Payload (version 1.0)	Type	Required
"isoFmd": false,	boolean	
"ansiFid": false,	boolean	
"ansiFmd": false,	boolean	
"qso": false,	boolean	
"wso": false,	boolean	
"reservedRaw1": false,	boolean	
"reservedTemplate1": false,	boolean	
"reservedRaw2": false,	boolean	
"reservedTemplate2": false,	boolean	
"reservedRaw3": false,	boolean	
"reservedTemplate3": false	boolean	
},		
"encryptionalAlgorithm": {	object	
"ecb": false,	boolean	
"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": "storedMatch",	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	
}		
},		

Payload (version 1.0)	Type	Required
"camera": {	object	
"type": "cam",	string	
"cameras": {	object	
"room": false,	boolean	
"person": false,	boolean	
"exitSlot": false,	boolean	
"*vendorSpecificCamera*": false	boolean	
},		
"maxPictures": 0,	integer	
"camData": {	object	
"autoAdd": false,	boolean	
"manAdd": false	boolean	
},		
"maxDataLength": 0,	integer	
"pictureFile": false	boolean	
},		
"lights": {	object	
"cardReader": {	object	
"flashRate": {	object	
"off": false,	boolean	
"slow": false,	boolean	
"medium": false,	boolean	
"quick": false,	boolean	
"continuous": false	boolean	
},		
"color": {	object	
"red": false,	boolean	
"green": false,	boolean	
"yellow": false,	boolean	
"blue": false,	boolean	
"cyan": false,	boolean	
"magenta": false,	boolean	
"white": false	boolean	
},		
"direction": {	object	
"entry": false,	boolean	
"exit": false	boolean	
},		
"position": {	object	
"left": false,	boolean	
"right": false,	boolean	

Payload (version 1.0)	Type	Required
"center": false,	boolean	
"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.		
},		
"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	

Payload (version 1.0)	Type	Required
See cardReader properties.		
},		
"cardReader2": {	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.		
},		
"fasciaLight": {	object	
See cardReader properties.		
},		
"*vendorSpecificLight*": {	object	
See cardReader properties.		
}		
},		
"auxiliaries": {	object	
"operatorSwitch": {	object	
"run": false,	boolean	
"maintenance": false,	boolean	
"supervisor": false	boolean	
},		
"tamperSensor": false,	boolean	
"internalTamperSensor": false,	boolean	

Payload (version 1.0)	Type	Required
"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	
"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	

Payload (version 1.0)	Type	Required
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	
"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	

Payload (version 1.0)	Type	Required
"basic": false,	boolean	
"intermediate": false,	boolean	
"full": false,	boolean	
"accessNotSupported": false	boolean	
}		
}		
}		

Properties

completionCode
The completion code.

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

interfaces
Array of interfaces supported by this XFS4IoT service.

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.
- `BarcodeReader` - BarcodeReader interface.
- `Lights` - Lights interface.
- `Auxiliaries` - Auxiliaries interface.
- `VendorMode` - VendorMode interface.
- `VendorApplication` - VendorApplication interface.
- `Storage` - Storage 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.

Properties
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 0 if the maximum number of requests is unlimited. Property value constraints: `minimum: 0`
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.

Properties
common/powerSaveControl Specifies whether power saving control is available.
common/antiFraudModule Specifies whether the anti-fraud module is available.
common/endToEndSecurity If this value is present then end to end security is supported. The sub-properties detail exactly how it is supported and what level of support is enabled. Also see common.StatusProperties.endToEndSecurity for the current status of end to end security, such as if it is being enforced, or if configuration is required. If this value is not present then end to end security is not supported by this service.
common/endToEndSecurity/required Specifies the level of support for end to end security • `ifConfigured` - The device is capable of supporting E2E security, but it will not be enforced if not configured, for example because the required keys are not loaded. • `always` - E2E security is supported and enforced at all times. Failure to supply the required security details will lead to errors. If E2E security is not correctly configured, for example because the required keys are not loaded, all secured commands will fail with an error. If end to end security is not supported this value will not be present.
common/endToEndSecurity/hardwareSecurityElement Specifies if this device has a Hardware Security Element (HSE) which validates the security token. If this property is false it means that validation is performed in software.
common/endToEndSecurity/responseSecurityEnabled Specifies 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. • `notSupported` - The device is incapable of returning a response token. • `ifConfigured` - The device is capable of supporting E2E security if correctly configured. If E2E security has not been correctly configured, for example because the required keys are not loaded, commands will complete without a security token. • `always` - A security token will be included with command responses. If E2E security is not correctly configured, for example because the required keys are not loaded, the command will complete with an error.
common/endToEndSecurity/commands Array of commands which require an E2E token to authorize. These commands will fail if called without a valid token. The commands that can be listed here depends on the XFS4IoT standard, but it's possible that the standard will change over time, so for maximum compatibility an application should check this property before calling a command. Note that this only includes commands that *require* a token. Commands that take a nonce and *return* a token will not be listed here. Those commands can be called without a nonce and will continue to operate in a compatible way. Property value constraints: `pattern: ^[A-Za-z][A-Za-z0-9]*\.[A-Za-z][A-Za-z0-9]*$`

Properties
common/endToEndSecurity/commandNonceTimeout If this device supports end to end security and can return a command nonce with the command Common.GetCommandNonce, and the device 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. The device 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 device. For example, a command nonce might time out after one hour (that is, 3600 seconds). In all other cases, commandNonceTimeout will have a value of zero. Any command nonce will never timeout. It may still become invalid, for example because of a power failure or when explicitly cleared using the ClearCommandNonce command. Property value constraints: `minimum: 0`
cardReader Capability information for XFS4IoT services implementing the CardReader interface. This will be omitted if the CardReader interface is not supported.
cardReader/type Specifies the type of the ID card unit as one of the following: • `motor` - The ID card unit is a motor driven card unit. • `swipe` - The ID card unit is a swipe (pull-through) card unit. • `dip` - The ID card unit is a dip card unit. This dip type is not capable of latching cards entered. • `latchedDip` - The ID card unit is a latched dip card unit. This device type is used when a dip card unit device supports chip communication. The latch ensures the consumer cannot remove the card during chip communication. Any card entered will automatically latch when a request to initiate a chip dialog is made (via the CardReader.ReadRawData command). The CardReader.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 [Ref. cardreader-3] 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.

Properties
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 written by the card reader.
cardReader/writeTracks/track1 The card reader can write on track 1.
cardReader/writeTracks/track2 The card reader can write on track 2.
cardReader/writeTracks/track3 The card reader can write on track 3.
cardReader/writeTracks/frontTrack1 The card reader can write on front track 1.
cardReader/writeTracks/track1JIS The card reader can write on JIS I track 1.
cardReader/writeTracks/track3JIS The card reader can write on 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=1 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.

Properties
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 be 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.
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 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.
cardReader/writeMode/loco Supports writing of loco magnetic stripes.
cardReader/writeMode/hico Supports writing of hico magnetic stripes.
cardReader/writeMode/auto The Service 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.
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:
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.Move 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.

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.MediaRemovedEvent 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. The 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 CashManagement.OpenShutter and CashManagement.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 CashManagement.OpenShutter and CashManagement.CloseShutter commands. In either case the CashAcceptor.PresentMedia command may be used if the *presentControl* property is false. This property 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. Omitted if there is no intermediate stacker for cash-in available. Property value constraints: `minimum: 0`
cashAcceptor/itemsTakenSensor Specifies whether the CashAcceptor can detect when items at the exit position are taken by the user. If true the Service generates an accompanying CashManagement.ItemsTakenEvent. If false this event is not generated. This property relates to all output positions.
cashAcceptor/itemsInsertedSensor Specifies whether the CashAcceptor has the ability to detect when items have been inserted by the user. If true the service generates an accompanying CashManagement.ItemsInsertedEvent. If false this event is not generated. This relates to all input positions and should not be reported as true unless item insertion can be detected.
cashAcceptor/positions Specifies the CashAcceptor input and output positions which are available.

Properties
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 storage unit.
cashAcceptor/retractAreas/transport The items may be retracted to the transport.
cashAcceptor/retractAreas/stacker The items may be retracted to the intermediate stacker.
cashAcceptor/retractAreas/reject The items may be retracted to a reject storage unit.
cashAcceptor/retractAreas/billCassette The items may be retracted to cash-in and recycle storage units.
cashAcceptor/retractAreas/cashIn Items may be retracted to cash-in storage units.

Properties
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 storage unit.
cashAcceptor/retractTransportActions/reject The items may be moved to a reject storage unit.
cashAcceptor/retractTransportActions/billCassette The items may be moved to the cash-in and recycle storage units.
cashAcceptor/retractTransportActions/cashIn Items may be retracted to cash-in storage units.
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 storage unit.
cashAcceptor/retractStackerActions/reject The items may be moved to a reject storage unit.
cashAcceptor/retractStackerActions/billCassette The items may be moved to the cash-in and recycle storage units.
cashAcceptor/retractStackerActions/cashIn Items may be retracted to cash-in storage units.
cashAcceptor/cashInLimit Specifies which cash-in limitations are supported 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 this property is omitted.
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 this property is omitted.
cashAcceptor/countActions/individual The counting of individual storage units is supported.
cashAcceptor/countActions/all The counting of all storage units is supported.

Properties
cashAcceptor/counterfeitAction Specifies whether counterfeit or suspect items (see Note Classification) are allowed to be returned to the customer during a cash-in transaction. If items 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. The following rules are possible: • `none` - The device is not able to classify items as counterfeit or suspect. • `level2` - Items are classified including counterfeit or suspect. Counterfeit items will not be returned to the customer in a cash-in transaction. • `level23` - Items are classified including counterfeit or suspect. Counterfeit and suspect items will not be returned to the customer in a cash-in transaction.
cashDispenser Capability information for XFS4IoT services implementing the CashDispenser interface. This will be omitted if the CashDispenser interface is not supported.
cashDispenser/type Supplies the type of Dispenser. Following values are possible: • `tellerBill` - The Dispenser is a Teller Bill Dispenser. • `selfServiceBill` - The Dispenser is a Self-Service Bill Dispenser. • `tellerCoin` - The Dispenser is a Teller Coin Dispenser. • `selfServiceCoin` - The Dispenser is a Self-Service Coin Dispenser.
cashDispenser/maxDispenseItems Supplies the maximum number of items that can be dispensed in a single dispense operation. Property value constraints: `minimum: 1`
cashDispenser/shutterControl If true the shutter is controlled implicitly by the Service. If false the shutter must be controlled explicitly by the application using the CashManagement.OpenShutter and CashManagement.CloseShutter commands. This property is always true if the device has no shutter. This property 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 storage 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 storage unit.
cashDispenser/retractAreas/itemCassette The items may be retracted to storage units which would be used during a Cash In transaction including recycling storage units.
cashDispenser/retractAreas/cashIn The items may be retracted to storage units which would be used during a Cash In transaction not including recycling storage units.
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.

Properties
cashDispenser/retractTransportActions/present The items may be presented.
cashDispenser/retractTransportActions/retract The items may be moved to a retract storage unit.
cashDispenser/retractTransportActions/reject The items may be moved to a reject storage unit.
cashDispenser/retractTransportActions/itemCassette The items may be moved to storage units which would be used during a Cash In transaction including recycling storage units.
cashDispenser/retractTransportActions/cashIn The items may be moved to storage units which would be used during a Cash In transaction not including recycling storage units.
cashDispenser/retractStackerActions Specifies the actions which may be performed on items which have been retracted to the stacker. If the device does not have the capability to retract items to the stacker or move items from the stacker all flags will be set to false.
cashDispenser/retractStackerActions/present The items may be presented.
cashDispenser/retractStackerActions/retract The items may be moved to a retract storage unit.
cashDispenser/retractStackerActions/reject The items may be moved to a reject storage unit.
cashDispenser/retractStackerActions/itemCassette The items may be moved to storage units which would be used during a Cash In transaction including recycling storage units.
cashDispenser/retractStackerActions/cashIn The items may be moved to storage units which would be used during a Cash In transaction not including recycling storage units.
cashDispenser/intermediateStacker Specifies whether the Dispenser supports stacking items to an intermediate position before the items are moved to the exit position.
cashDispenser/itemsTakenSensor Specifies whether the Dispenser can detect when items at the exit position are taken by the user. This applies to all output positions. If true the Service generates an accompanying CashManagement.ItemsTakenEvent. If false this event is not generated.
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.

Properties
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 storage units to the intermediate stacker while there are items on the transport.
cashDispenser/moveItems/toCashUnit The Dispenser can retract items to the storage units while there are items on the intermediate stacker.
cashDispenser/moveItems/toTransport The Dispenser can retract items to the transport while there are items on the intermediate stacker.
cashDispenser/moveItems/toStacker The Dispenser can dispense items from the storage 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.
cashManagement Capability information for XFS4IoT services implementing the CashManagement interface. This will be omitted if the CashManagement interface is not supported.
cashManagement/cashBox This property 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 storage unit exchange operations supported by the device.
cashManagement/exchangeType/byHand The device supports manual replenishment either by filling the storage unit by hand or by replacing the storage unit.
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/image Image of the item.
cashManagement/classificationList Specifies whether the Service has the capability to maintain a classification list of serial numbers as well as supporting the associated operations.
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.

Properties
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/banksys PIN is encrypted and formatted according to the Banksys PIN block specifications.
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.
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 (see [Ref. pinpad-5]). 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/pinFormats/iso4 PIN is formatted according to ISO 9564-1: 2017 Format-4 (uses AES Encryption).
pinPad/presentationAlgorithms Supported presentation algorithms.

Properties
pinPad/presentationAlgorithms/presentClear Algorithm for the presentation of a clear text PIN to a chipcard. Each digit of the clear text PIN is inserted as one nibble (=halfbyte) into ChipData.
pinPad/display Specifies the type of the display used in the PIN pad module.
pinPad/display/none No display unit.
pinPad/display/ledThrough Lights next to text guide user.
pinPad/display/display A real display is available (this doesn't apply for self-service).
pinPad/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.
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$`
pinPad/pinBlockAttributes/P0/T (example name) Specifies the encryption algorithms supported by the PinPad.PinBlock command. The following values are possible: • `A` - AES. • `D` - DEA. • `R` - RSA. • `T` - Triple DEA (also referred to as TDEA). Property name constraints: `pattern: ^[ADRT]$`

Properties
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 (See [Ref. pinpad-7]). • `xts` - The XTS method defined in NIST SP800-38E (See [Ref. pinpad-8]). 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 OFB encryption method.
pinPad/pinBlockAttributes/P0/T/E/cryptoMethod/ctr The CTR method defined in NIST SP800-38A (See [Ref. pinpad-7]).
pinPad/pinBlockAttributes/P0/T/E/cryptoMethod/xts The XTS method defined in NIST SP800-38E (See [Ref. pinpad-8]).
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 [Ref. crypto-1] and FIPS 180-2 [Ref. crypto-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.

Properties
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]$`
crypto/cryptoAttributes/D0/D (example name) Specifies the encryption algorithms supported by the Crypto.CryptoData command. The following values are possible: • `A` - AES. • `D` - DEA. • `R` - RSA. • `T` - Triple DEA (also referred to as TDEA). Property name constraints: `pattern: ^[ADRT]$`
crypto/cryptoAttributes/D0/D/D (example name) Specifies the Mode of Use supported by the Crypto.CryptoData command. The following values are possible: • `D` - Decrypt • `E` - Encrypt Property name constraints: `pattern: ^[DE]$`
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 (See [Ref. crypto-4]) • `xts` - The XTS method defined in NIST SP800-38E (See [Ref. crypto-5]) 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 OFB encryption method.
crypto/cryptoAttributes/D0/D/D/cryptoMethod/ctr The CTR method defined in NIST SP800-38A (See [Ref. crypto-4])
crypto/cryptoAttributes/D0/D/D/cryptoMethod/xts The XTS method defined in NIST SP800-38E. (See [Ref. crypto-5])
crypto/cryptoAttributes/D0/D/D/cryptoMethod/rsaesPkcs1V15 The RSAES_PKCS1-v1.5 algorithm.

Properties
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.
crypto/authenticationAttributes/M0 (example name) The following key usages 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` - 9797-1:2011 MAC Algorithm 5/CMAC. • `M7` - HMAC. • `M8` - ISO 9797-1:2011 MAC Algorithm 6. • `S0` - Asymmetric key pair for digital signature. • `S1` - Asymmetric key pair, CA. • `S2` - Asymmetric key pair, nonX9.24 key. Property name constraints: `pattern: ^M[0-8]$\|^S[0-2]$`
crypto/authenticationAttributes/M0/T (example name) Specifies the encryption algorithms supported by the [Crypto.GenerateAuthentication](#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]$`
crypto/authenticationAttributes/M0/T/G (example name) Specifies the Mode of Use supported by the Crypto.GenerateAuthentication command. The following values are possible: • `C` - Both Generate and Verify. • `G` - Generate. This be used to generate a MAC. • `S` - Signature • `T` - Both Sign and Decrypt. Property name constraints: `pattern: ^[CGST]$`
crypto/authenticationAttributes/M0/T/G/cryptoMethod Specifies the asymmetric signature verification method supported by the Crypto.GenerateAuthentication command. If the key usage is one of the MAC usages (e.g. 'M0'), the following properties are false.
crypto/authenticationAttributes/M0/T/G/cryptoMethod/rsassaPkcs1V15 The RSASSA-PKCS1-v1.5 algorithm.
crypto/authenticationAttributes/M0/T/G/cryptoMethod/rsassaPss The RSASSA-PSS algorithm.
crypto/authenticationAttributes/M0/T/G/hashAlgorithm Specifies the hash algorithm supported. If the *key* usage is one of the MAC usages (e.g. 'M0'), the following properties are false.

Properties
crypto/authenticationAttributes/M0/T/G/hashAlgorithm/sha1 The SHA 1 digest algorithm.
crypto/authenticationAttributes/M0/T/G/hashAlgorithm/sha256 The SHA 256 digest algorithm, as defined in ISO/IEC 10118-3:2004 [Ref. crypto-1] and FIPS 180-2 [Ref. crypto-2].
crypto/verifyAttributes Key-value pair of attributes supported by the Crypto.VerifyAuthentication command to verify authentication data.
crypto/verifyAttributes/M0 (example name) The following key usages 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` - 9797-1:2011 MAC Algorithm 5/CMAC. • `M7` - HMAC. • `M8` - ISO 9797-1:2011 MAC Algorithm 6. • `S0` - Asymmetric key pair for digital signature. • `S1` - Asymmetric key pair, CA. • `S2` - Asymmetric key pair, nonX9.24 key. Property name constraints: `pattern: ^M[0-8]$\|^S[0-2]$`
crypto/verifyAttributes/M0/T (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]$`
crypto/verifyAttributes/M0/T/V (example name) Specifies the Mode of Use supported by Crypto.VerifyAuthentication command. The following values are possible: • `V` - Verify. This be used to verify a MAC. Property name constraints: `pattern: ^V$`
crypto/verifyAttributes/M0/T/V/cryptoMethod Specifies the asymmetric signature verification method supported by the Crypto.VerifyAuthentication command. If the key usage is one of the MAC usages (e.g. 'M0'), the following properties are false.
crypto/verifyAttributes/M0/T/V/cryptoMethod/rsassaPkcs1V15 The RSASSA-PKCS1-v1.5 algorithm.
crypto/verifyAttributes/M0/T/V/cryptoMethod/rsassaPss The RSASSA-PSS algorithm.
crypto/verifyAttributes/M0/T/V/hashAlgorithm Specifies the hash algorithm supported. If the key usage is one of the MAC usages (e.g. 'M0'), the following properties are false.

Properties
crypto/verifyAttributes/M0/T/V/hashAlgorithm/sha1 The SHA 1 digest algorithm.
crypto/verifyAttributes/M0/T/V/hashAlgorithm/sha256 The SHA 256 digest algorithm, as defined in ISO/IEC 10118-3:2004 [Ref. crypto-1] and FIPS 180-2 [Ref. crypto-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. This should be omitted if not supported or the HSM vendor is unknown.
keyManagement/rsaAuthenticationScheme Specifies the types of Remote Key Loading/Authentication that are supported.
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 TR34-2019 [Ref. keymanagement-9].
keyManagement/rsaSignatureAlgorithm Specifies the types of RSA Signature Algorithm that are supported.
keyManagement/rsaSignatureAlgorithm/pkcs1V15 pkcs1V15 Signatures supported.
keyManagement/rsaSignatureAlgorithm/pss pss Signatures supported.
keyManagement/rsaCryptAlgorithm Specifies the types of RSA Encipherment Algorithm that are supported.
keyManagement/rsaCryptAlgorithm/pkcs1V15 pkcs1V15 algorithm supported.
keyManagement/rsaCryptAlgorithm/oaep oaep algorithm supported.

Properties
keyManagement/rsaKeyCheckMode Specifies which hash algorithms used to generate the public key check value/thumb print are supported.
keyManagement/rsaKeyCheckMode/sha1 sha1 is supported as defined in [Ref. keymanagement-2].
keyManagement/rsaKeyCheckMode/sha256 sha256 is supported as defined in ISO/IEC 10118-3:2004 [Ref. keymanagement-7] and FIPS 180-2 [Ref. keymanagement-8].
keyManagement/signatureScheme Specifies which capabilities are supported by the Signature scheme.
keyManagement/signatureScheme/randomNumber Specifies if the service returns a random number from the KeyManagement.StartKeyExchange command within the RSA Signature Scheme.
keyManagement/signatureScheme/exportDeviceId Specifies if the service supports exporting the device Security Item within the RSA Signature Scheme.
keyManagement/signatureScheme/enhancedRkl Specifies that the service 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 PK_{ROOT}, rather than PK_{HOST}.
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. See [Ref. keymanagement-3].
keyManagement/emvImportSchemes/epiCA A CA public key is imported using the selfsign scheme defined in the Europay International, EPI CA Module Technical - Interface specification Version 1.4, [Ref. ref-keymanagement-4].
keyManagement/emvImportSchemes/issuer An Issuer public key is imported as defined in EMV 2000 Book II, [Ref. keymanagement-3].
keyManagement/emvImportSchemes/icc An ICC public key is imported as defined in EMV 2000 Book II, [Ref. keymanagement-3].
keyManagement/emvImportSchemes/iccPin An ICC PIN public key is imported as defined in EMV 2000 Book II, [Ref. keymanagement-3].
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/A Supports X9.143 key block version ID A.
keyManagement/keyBlockImportFormats/B Supports X9.143 key block version ID B.
keyManagement/keyBlockImportFormats/C Supports X9.143 key block version ID C.

Properties
keyManagement/keyBlockImportFormats/D Supports X9.143 key block version ID D.
keyManagement/keyImportThroughParts Specifies whether the device is capable of importing keys in multiple parts.
keyManagement/desKeyLength Specifies which DES key lengths 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 device public encryption certificate.
keyManagement/certificateTypes/verificationKey Supports the device public verification certificate.
keyManagement/certificateTypes/hostKey Supports the Host public certificate.
keyManagement/loadCertOptions Specifying the options supported by the KeyManagement.LoadCertificate command.
keyManagement/loadCertOptions/signer Specifies the signers supported by the KeyManagement.LoadCertificate command. The possible values 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 TR34-2019 [Ref. keymanagement-9]. • `caTr34` - The Certificate Authority RSA Private Key is used to sign the token, compliant with X9 TR34-2019 [Ref. keymanagement-9]. • `hlTr34` - A Higher-Level Authority RSA Private Key is used to sign the token, compliant with X9 TR34-2019 [Ref. keymanagement-9].
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 (or rebind) the device 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.

Properties
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 [Ref. keymanagement-10]).
keyManagement/keyAttributes Key-value pair of attributes supported by KeyManagement.ImportKey command for the key to be loaded.

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.
- `B3` - Key Derivation Key (Non ANSI X9.24).
- `C0` - CVK Card Verification Key.
- `D0` - Symmetric Key for Data Encryption.
- `D1` - Asymmetric Key for Data Encryption.
- `D2` - Data Encryption Key for Decimalization Table.
- `D3` - Data Encryption Key for Sensitive Data.
- `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).
- `E7` - EMV / Chip Asymmetric Key Pair for EMV/Smart Card based PIN/PIN Block Encryption.
- `I0` - Initialization Vector (IV).
- `K0` - Key Encryption or wrapping.
- `K1` - X9.143 Key Block Protection Key.
- `K2` - TR-34 Asymmetric Key.
- `K3` - Asymmetric Key for key agreement / key wrapping.
- `K4` - Key Block Protection Key, ISO 20038.
- `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.
- `P1` - PIN Generation Key (reserved for ANSI X9.132-202x).
- `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.
- `V5` - PIN Verification Key, ANSI X9.132 algorithm 3.
- `00 – 99` - These numeric values are reserved for proprietary use.

Property name constraints:

```
pattern: ^B[0-3]$|^C0$|^D[0-3]$|^E[0-7]$|^I0$|^K[0-4]$|^M[0-8]$|^P[0-1]$|^S[0-
2]$|^V[0-5]$|^[0-9][0-9]$
```

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

Properties

keyManagement/keyAttributes/M0/T/C/restrictedKeyUsage

If the key usage is a key encryption usage (e.g. 'K0') this specifies the key usage of the keys that can be encrypted by the key.

This property should be omitted if restricted key usage is not supported or required..

The following values are possible:

- `B0` - BDK Base Derivation Key.
- `B1` - Initial DUKPT key.
- `B2` - Base Key Variant Key.
- `B3` - Key Derivation Key (Non ANSI X9.24).
- `C0` - CVK Card Verification Key.
- `D0` - Symmetric Key for Data Encryption.
- `D1` - Asymmetric Key for Data Encryption.
- `D2` - Data Encryption Key for Decimalization Table.
- `D3` - Data Encryption Key for Sensitive Data.
- `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).
- `E7` - EMV / Chip Asymmetric Key Pair for EMV/Smart Card based PIN/PIN Block Encryption.
- `I0` - Initialization Vector (IV).
- `K0` - Key Encryption or wrapping.
- `K1` - X9.143 Key Block Protection Key.
- `K2` - TR-34 Asymmetric Key.
- `K3` - Asymmetric Key for key agreement / key wrapping.
- `K4` - Key Block Protection Key, ISO 20038.
- `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.
- `P1` - PIN Generation Key (reserved for ANSI X9.132-202x).
- `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.
- `V5` - PIN Verification Key, ANSI X9.132 algorithm 3.
- `00 – 99` - These numeric values are reserved for proprietary use.

Property value constraints:

```
pattern: ^B[0-3]$|^C0$|^D[0-3]$|^E[0-7]$|^I0$|^K[0-4]$|^M[0-8]$|^P[0-1]$|^S[0-2]$|^V[0-5]$|^[0-9][0-9]$
```

Properties
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.
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 (See [Ref. keymanagement-11]). • `xts` - The XTS method defined in NIST SP800-38E (See [Ref. keymanagement-12]). 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 OFB encryption method.
keyManagement/decryptAttributes/A/decryptMethod/ctr The CTR method defined in NIST SP800-38A (See [Ref. 11]).
keyManagement/decryptAttributes/A/decryptMethod/xts The XTS method defined in NIST SP800-38E (See [Ref. keymanagement-12]).
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.

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]$
```

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.

Properties
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 both 'sha1' and 'sha256' properties 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 [Ref. keymanagement-7] and FIPS 180-2 [Ref. keymanagement-8].
keyboard Capability information for XFS4IoT services implementing the Keyboard interface. This will be omitted if the Keyboard interface is not supported.
keyboard/autoBeep Specifies whether the device will emit a key beep tone on key presses of active keys or inactive keys, and if so, which mode it supports.
keyboard/autoBeep/activeAvailable Automatic beep tone on active key key-press is supported. If this flag is not set then automatic beeping for active keys is not supported.
keyboard/autoBeep/activeSelectable Automatic beeping for active keys can be controlled turned on and off by the 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 inactive key keypress is supported. If this flag is not set then automatic beeping for inactive keys is not supported.
keyboard/autoBeep/inactiveSelectable Automatic beeping for inactive keys can be controlled turned on and off by the application. If this flag is not set then automatic beeping for inactive keys cannot be controlled by an application.
keyboard/etsCaps Specifies the capabilities of the Encrypting Touch Screen device.
keyboard/etsCaps/xPos Specifies the position of the left edge of the Encrypting Touch Screen in virtual screen coordinates. This value may be negative because the of the monitor position on the virtual desktop. Property value constraints: `minimum: 0`

Properties
keyboard/etsCaps/yPos Specifies the position of the right edge of the Encrypting Touch Screen in virtual screen coordinates. This value may be negative because the of the monitor position on the virtual desktop. Property value constraints: `minimum: 0`
keyboard/etsCaps/xSize Specifies the width of the Encrypting Touch Screen in virtual screen coordinates. Property value constraints: `minimum: 0`
keyboard/etsCaps/ySize Specifies the height of the Encrypting Touch Screen in virtual screen coordinates. Property value constraints: `minimum: 0`
keyboard/etsCaps/maximumTouchFrames Specifies the maximum number of Touch-Frames that the device can support in a touch keyboard definition. Property value constraints: `minimum: 0`
keyboard/etsCaps/maximumTouchKeys Specifies the maximum number of Touch-Keys that the device can support within a touch frame. Property value constraints: `minimum: 0`
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: • `fixed` - The text terminal unit is a fixed device. • `removable` - The text terminal unit is a removable device.
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 is initialized or reset through the TextTerminal.Reset command.
textTerminal/resolutions/sizeX Specifies 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`

Properties
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.
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.
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.

Properties
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.
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.
printer/extents/horizontal Device has horizontal size detection capability.
printer/extents/vertical Device has vertical size detection capability.
printer/control Specifies the manner in which media can be controlled.
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.

Properties
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 can clear buffered data.
printer/maxMediaOnStacker Specifies the maximum number of media items that the stacker can hold. Omitted if not available. Property value constraints: `minimum: 0`
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.
printer/paperSources/upper The upper paper source.
printer/paperSources/lower The lower paper source.
printer/paperSources/external The external paper source.
printer/paperSources/aux The auxiliary paper source.
printer/paperSources/aux2 The second auxiliary paper source.
printer/paperSources/park The parking station.

Properties
printer/paperSources/exampleProperty1 (example name) The vendor specific paper source. Property name constraints: `pattern: ^[a-zA-Z]([a-zA-Z0-9]*)$`
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. Omitted if not available. Property value constraints: `minimum: 0`
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 1 to *retractBins* - 1 for bin number *retractBins*). This will be omitted if there are no retract bins. Property value constraints: `minimum: 0`
printer/imageType Specifies the image format supported by this device.
printer/imageType/tif The device can return scanned images in TIFF 6.0 format.
printer/imageType/wmf The device can return scanned images in WMF (Windows Metafile) format.
printer/imageType/bmp The device can return scanned images in Windows BMP format.
printer/imageType/jpg The device can return scanned images in JPG format.
printer/frontImageColorFormat Specifies the front image color formats supported by this device.
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.
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.

Properties
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.
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. Applications should use this printer name to generate native printer files to be printed through the Printer.PrintNative command. This value will be omitted if the Service does not support the *Printer.PrintNative* command.
printer/mediaPresented Specifies whether the device is able to detect when the media is presented to the user for removal. If true, the Printer.MediaPresentedEvent event is fired. If false, the Printer.MediaPresentedEvent event is not fired.
printer/autoRetractPeriod Specifies the number of seconds before the device will automatically retract the presented media. If the command that generated the media is still active when the media is automatically retracted, the command will complete with an error. Omitted if the device does not retract media automatically.
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.
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 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.
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.

Properties
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. • `notSupported` - 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.
barcodeReader Capability information for XFS4IoT services implementing the BarcodeReader interface. This will be omitted if the BarcodeReader interface is not supported.
barcodeReader/canFilterSymbologies Specifies whether the device is capable of discriminating between the presented barcode symbologies such that only the desired symbologies are recognized/reported
barcodeReader/symbologies Specifies the barcode symbologies readable by the scanner. This will be omitted if the supported barcode symbologies can not be determined.
barcodeReader/symbologies/ean128 GS1-128
barcodeReader/symbologies/ean8 EAN-8
barcodeReader/symbologies/ean8_2 EAN-8 with 2 digit add-on
barcodeReader/symbologies/ean8_5 EAN-8 with 5 digit add-on
barcodeReader/symbologies/ean13 EAN-13
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

Properties
barcodeReader/symbologies/upcE1_5 UPC-E with leading 1and 5 digit add-on
barcodeReader/symbologies/upcA_2 UPC-A with2 digit add-on
barcodeReader/symbologies/upcA_5 UPC-A with 5 digit add-on
barcodeReader/symbologies/codabar CODABAR (NW-7)
barcodeReader/symbologies/itf Interleaved 2 of 5 (ITF)
barcodeReader/symbologies/code11 CODE 11 (USD-8)
barcodeReader/symbologies/code39 CODE 39
barcodeReader/symbologies/code49 CODE 49
barcodeReader/symbologies/code93 CODE 93
barcodeReader/symbologies/code128 CODE 128
barcodeReader/symbologies/msi MSI
barcodeReader/symbologies/plessey PLESSEY
barcodeReader/symbologies/std2Of5 STANDARD 2 of 5 (INDUSTRIAL 2 of 5 also)
barcodeReader/symbologies/std2Of5Iata STANDARD 2 of 5 (IATA Version)
barcodeReader/symbologies/pdf417 PDF-417
barcodeReader/symbologies/microPdf417 MICROPDF-417
barcodeReader/symbologies/dataMatrix GS1 DataMatrix
barcodeReader/symbologies/maxiCode MAXICODE
barcodeReader/symbologies/codeOne CODE ONE
barcodeReader/symbologies/channelCode CHANNEL CODE
barcodeReader/symbologies/telepenOriginal Original TELEPEN
barcodeReader/symbologies/telepenAim AIM version of TELEPEN

Properties
barcodeReader/symbologies/rss GS1 DataBar™
barcodeReader/symbologies/rssExpanded Expanded GS1 DataBar™
barcodeReader/symbologies/rssRestricted Restricted GS1 DataBar™
barcodeReader/symbologies/compositeCodeA Composite Code A Component
barcodeReader/symbologies/compositeCodeB Composite Code B Component
barcodeReader/symbologies/compositeCodeC Composite Code C Component
barcodeReader/symbologies/posiCodeA Posicode Variation A
barcodeReader/symbologies/posiCodeB Posicode Variation B
barcodeReader/symbologies/triopticCode39 Trioptic Code 39
barcodeReader/symbologies/codablockF Codablock F
barcodeReader/symbologies/code16K Code 16K
barcodeReader/symbologies/qrCode QR Code
barcodeReader/symbologies/aztec Aztec Codes
barcodeReader/symbologies/ukPost UK Post
barcodeReader/symbologies/planet US Postal Planet
barcodeReader/symbologies/postnet US Postal Postnet
barcodeReader/symbologies/canadianPost Canadian Post
barcodeReader/symbologies/netherlandsPost Netherlands Post
barcodeReader/symbologies/australianPost Australian Post
barcodeReader/symbologies/japanesePost Japanese Post
barcodeReader/symbologies/chinesePost Chinese Post
barcodeReader/symbologies/koreanPost Korean Post

Properties
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.
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. If this is zero then the device or the Service determines how many captures will be attempted. Property value constraints: `minimum: 0`
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. Property value constraints: `minimum: 0`
biometric/dataFormats Specifies the supported biometric raw data and template data formats reported.
biometric/dataFormats/isoFid Raw ISO FID format [Ref. biometric-3].
biometric/dataFormats/isoFmd ISO FMD template format [Ref. biometric-4].
biometric/dataFormats/ansiFid Raw ANSI FID format [Ref. biometric-1].

Properties
biometric/dataFormats/ansiFmd ANSI FMD template format [Ref. biometric-2].
biometric/dataFormats/qso Raw QSO image format.
biometric/dataFormats/wso WSQ image format.
biometric/dataFormats/reservedRaw1 Reserved for a vendor-defined Raw format.
biometric/dataFormats/reservedTemplate1 Reserved for a vendor-defined Template format.
biometric/dataFormats/reservedRaw2 Reserved for a vendor-defined Raw format.
biometric/dataFormats/reservedTemplate2 Reserved for a vendor-defined Template format.
biometric/dataFormats/reservedRaw3 Reserved for a vendor-defined Raw format.
biometric/dataFormats/reservedTemplate3 Reserved for a vendor-defined Template format.
biometric/encryptionalAlgorithm Supported encryption algorithms. Omitted 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. This applies specifically to the biometric data that has been captured using the Biometric.Read. 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 can persist until all sessions are closed, the device is power failed or rebooted, or the Biometric.Readis requested again. This captured biometric data can also be explicitly cleared using the Biometric.Clear or Biometric.Reset.
biometric/persistenceModes/clear Captured biometric data will not persist. Once the data has been either returned in the Biometric.Reador used by the Biometric.Match, then the data is cleared from the device.

Properties
biometric/matchSupported Specifies if matching is supported using the Biometric.Matchand/or Biometric.SetMatch command. Omitted if the device does not support matching. This will be one of the following values: • `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. • `combinedMatch` - The device scans biometric data and performs a match against imported biometric data as a single operation. The Biometric.SetMatchmust be called before the Biometric.Read in order to set the matching criteria. Then the Biometric.Match can be called to return the result.
biometric/scanModes Specifies the scan modes that can be used through the Biometric.Read.
biometric/scanModes/scan The Biometric.Read 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 can be used to scan data for a match operation using the Biometric.Match.
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.Clearor Biometric.Reset command.
biometric/clearData/scannedData Raw image data that has been scanned using the Biometric.Read can be cleared.
biometric/clearData/importedData Template data that was imported using the Biometric.Import can be cleared.
biometric/clearData/setMatchedData Match criteria data that was set using the Biometric.Match 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. • `cam` - Camera system.
camera/cameras Specifies whether cameras are available.
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/cameras/vendorSpecificCamera (example name) Allows vendor specific cameras to be reported.

Properties
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. 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 Camera.TakePicture.camData.
camera/maxDataLength Specifies the maximum length of the data that is displayed on the photo. Omitted if data cannot be manually added to the picture. Property value constraints: `minimum: 0`
camera/pictureFile Specifies whether the parameter Camera.TakePicture.pictureFile is supported. supported.
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.
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.

Properties
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.
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/billAcceptor Bill Acceptor Light.
lights/envelopeDispenser Envelope Dispenser Light.

Properties
lights/documentPrinter Document Printer Light.
lights/coinAcceptor Coin Acceptor Light.
lights/scanner Scanner Light.
lights/contactless Contactless Reader Light.
lights/cardReader2 Card Reader 2 Light.
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/vendorSpecificLight (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

Properties
auxiliaries/internalTamperSensor 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
auxiliaries/enhancedAudioSensor Specifies which modes 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 modes the Handset supports, if present.
auxiliaries/handsetSensor/manual The Handset is available and it supports manual mode. default: false

Properties
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.
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.

Properties
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
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`

Properties
auxiliaries/ups Specifies what states the UPS device supports.
auxiliaries/ups/low The UPS can indicate that its charge level is low. default: false
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.
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 Auxiliaries.SetAuxiliaries. 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.
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 Auxiliaries.SetAuxiliaries. default: false

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

Properties
auxiliaries/autoStartupMode Specifies which modes of the auto start-up control are supported.
auxiliaries/autoStartupMode/specific The device supports one-time auto start-up on a specific date at a specific time. default: false
auxiliaries/autoStartupMode/daily The 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 property will be omitted. Otherwise this will report the supported access levels. 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

4.1.3 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 Mode.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"commands": {	object	
"*CardReader.ReadRawData*": 1,	integer	
"*CardReader.Move*": *1*	integer	
},		
"events": {	object	
"*CardReader.MediaInsertedEvent*": 1,	integer	
"*CardReader.MediaRemovedEvent*": *1*	integer	
}		
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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
```

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 (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

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

Event Messages

None

4.1.4 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 Mode.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"requestIds": [1, 2]	array (integer)	
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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 requestIds 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 (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "noMatchingRequestIDs"	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:

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

Event Messages

None

4.1.5 Common.PowerSaveControl

This command activates or deactivates the power-saving mode. If the Service receives another command while in power saving mode:

- If the command requires the device to be powered up while in power saving mode, the Service automatically exits the power saving mode, and executes the requested command.
- If the command does not require the device to be powered up while in power saving mode, the Service will not exit the power saving mode.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"maxPowerSaveRecoveryTime": 5	integer	
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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 set to 0 then the device will exit the power saving mode.

Property value constraints:

```
minimum: 0
```

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

Properties

completionCode

The completion code.

errorDescription

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

Event Messages

None

4.1.6 Common.SetTransactionState

This command allows the application to specify the transaction state, which the Service can then utilize in order to optimize performance. After receiving this command, this Service can perform the necessary processing to start or end the customer transaction. This command should be called for every Service that could be used in a customer transaction. The transaction state applies to every session.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"state": "active",	string	
"transactionID": "Example transaction ID"	string	
}		

Properties

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

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.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

Properties

completionCode
The completion code.

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

Event Messages

None

4.1.7 Common.GetTransactionState

This command can be used to get the transaction state.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

timeout

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

default: 0

Completion Message

Payload (version 1.0)	Type	Required
{		
"state": "active",	string	
"transactionID": "Example transaction ID"	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.

Event Messages

None

4.1.8 Common.GetCommandNonce

Get a nonce to be included in an Authorization 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 Authorization 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 Authorization 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 (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"commandNonce": "646169ECDD0E440C2CECC8DDD7C27C22",	string	
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

Properties

commandNonce
A nonce that should be included in the Authorization 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

4.1.9 Common.ClearCommandNonce

Clear the command nonce from the device. The command nonce is included in an Authorization 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 (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

Properties

completionCode
The completion code.

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

Event Messages

None

4.2 Unsolicited Messages

4.2.1 Common.StatusChangedEvent

This event reports that a change of common state has occurred. The new value of all properties which have changed value are included in the event payload. Any properties which have not changed state are omitted.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
"device": "online",	string	
"devicePosition": "inPosition",	string	
"powerSaveRecoveryTime": 0,	integer	
"antiFraudModule": "ok",	string	
"exchange": "notSupported",	string	
"endToEndSecurity": "enforced"	string	
}		

Properties

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 or breaking an interlock).
- `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.
- `starting` - The device is starting and performing whatever initialization is necessary. This can be reported after the connection is made but before the device is ready to accept commands. This must only be a temporary state, the Service must report a different state as soon as possible. If an error causes initialization to fail then the state should change to *hardwareError*.

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.

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 0 if either the power saving mode has not been activated or no power save control is supported.

Properties
antiFraudModule Specifies the state of the anti-fraud module if available. Following values are possible: • `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.
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.StatusChangedEvent 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"
endToEndSecurity Specifies the status of end to end security support on this device. Also see Common.CapabilityProperties.endToEndSecurity. • `notSupported` - E2E security is not supported by this hardware. Any command can be called without a token. • `notEnforced` - E2E security is supported by this hardware but it is not currently enforced, for example because required keys aren't loaded. It's currently possible to perform E2E commands without a token. • `notConfigured` - E2E security is supported but not correctly configured, for example because required keys aren't loaded. Any attempt to perform any command protected by E2E security will fail. • `enforced` - E2E security is supported and correctly configured. E2E security will be enforced. Calling E2E protected commands will only be possible if a valid token is given. default: "notSupported"

4.2.2 Common.ErrorEvent

This event reports that an error has occurred. In most cases, this is in addition to being reported via the error code that is returned as the command completion.

In order to supply the maximum information, these events should be sent as soon as an error is detected. In particular, if an error is detected during the processing of a command, then the event should be sent before the command completion message.

Unsolicited Message

Payload (version 1.0)	Type	Required
`{`		
`"eventId": "hardware",`	string	Yes
`"action": "reset",`	string	
`"vendorDescription": "An error occurred in position B."`	string	
`}`		

Properties

eventId

Specifies the type of error. Following values are possible:

- `hardware` - The error is a hardware error.
- `software` - The error is a software error.
- `user` - The error is a user error.
- `fraudAttempt` - Some devices are capable of identifying a malicious physical attack which attempts to defraud valuable information or media. In this circumstance, this is returned to indicate a fraud attempt has occurred.

action

The action required to manage the error. This can be omitted if no action is required. Possible values are:

- `reset` - Reset device to attempt recovery using a *Reset* command, but should not be used excessively due to the potential risk of damage to the device. Intervention is not required, although if repeated attempts are unsuccessful then *maintenance* may be reported.
- `softwareError` - A software error occurred. Contact software vendor.
- `configuration` - A configuration error occurred. Check configuration.
- `clear` - Recovery is not possible. A manual intervention for clearing the device is required. This value is typically returned when a hardware error has occurred which banking personnel may be able to clear without calling for technical maintenance, e.g. 'replace paper', or 'remove cards from retain bin'.
- `maintenance` - Recovery is not possible. A technical maintenance intervention is required. This value is only used for hardware errors and fraud attempts. This value is typically returned when a hardware error or fraud attempt has occurred which requires field engineer specific maintenance activity. A *Reset* command may be used to attempt recovery after intervention, but should not be used excessively due to the potential risk of damage to the device. Vendor Application may be required to recover the device.
- `suspend` - Device will attempt auto recovery and will advise any further action required via a Common.StatusChangedEvent or another *Common.ErrorEvent*.

vendorDescription

A vendor-specific description of the error. May be omitted if not applicable.

4.2.3 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.

Unsolicited Message

Payload (version 1.0)	Type	Required

{
  "reasonDescription": "Nonce cleared by timeout"
}
	string	

Properties
reasonDescription Optional text describing why the nonce was cleared. The value of this text should not be relied on.

5. Card Reader Interface

This chapter defines the Card Reader interface functionality and messages.

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)

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

5.1 General Information

5.1.1 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
cardreader-4	ISO 8583:1987 Bank card originated messages — Interchange message specifications — Content for financial transactions

5.1.2 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 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 command sequences, as well as the cardholder and client 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 client 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].

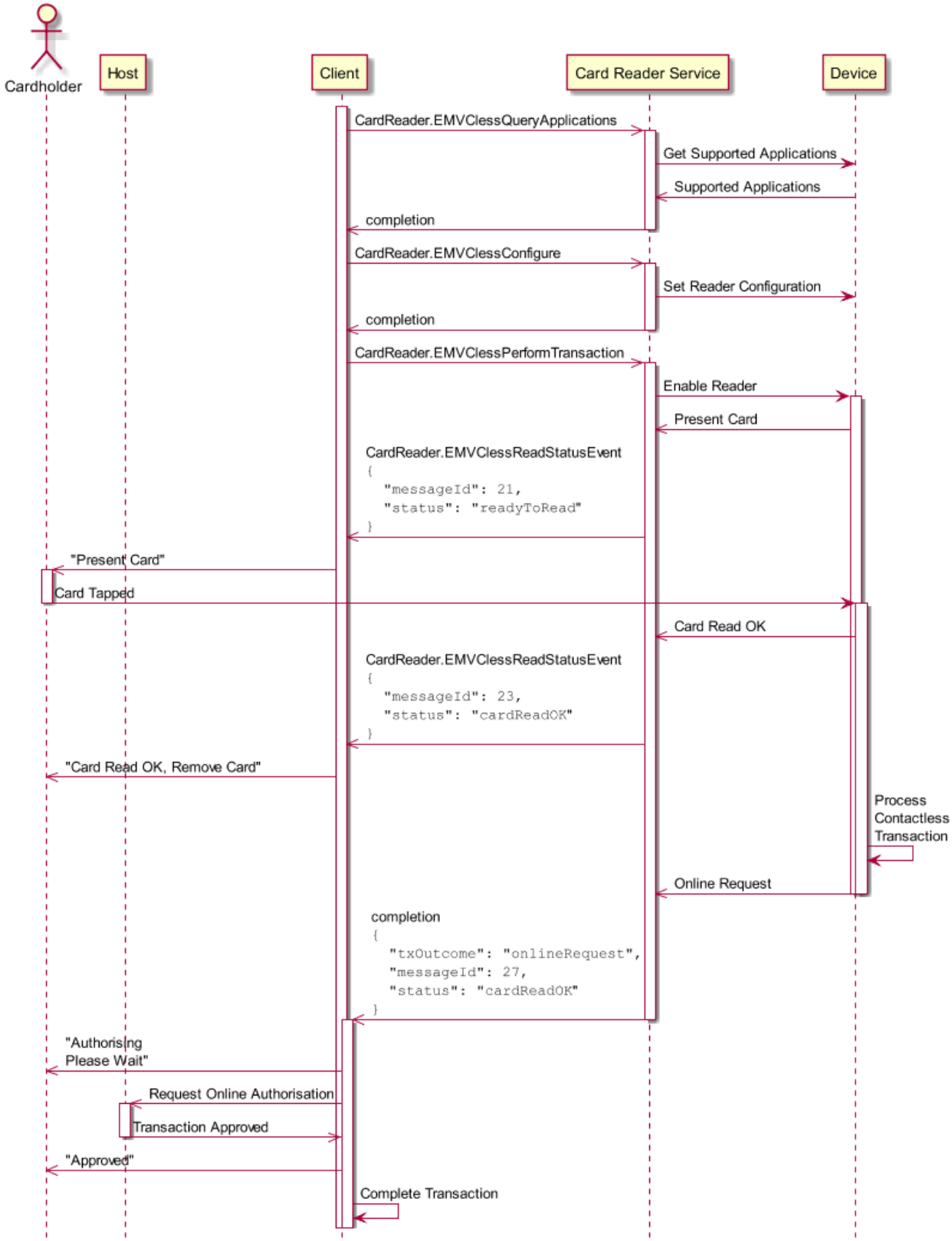
This document refers to BER-TLV tags. These are defined by each individual payment systems and contain the data exchanged between the client, contactless card and an intelligent contactless card reader. They are used to configure and prepare the intelligent contactless card reader for a transaction and are also part of the data that is returned by the reader on completion of a card tap.

Based on the applicable payment system 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.

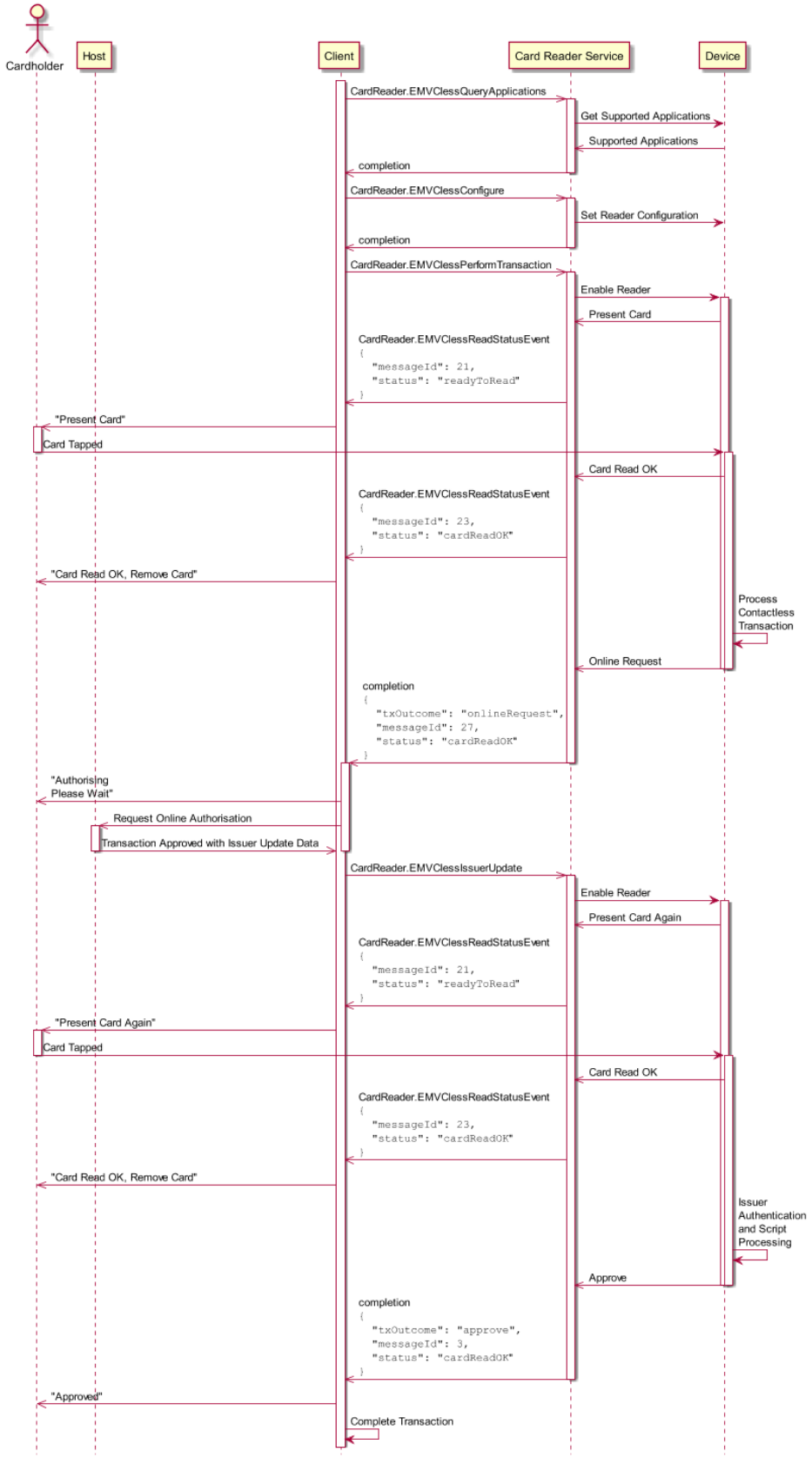
5.1.3 Intelligent Contactless Card Reader Sequence Diagrams

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

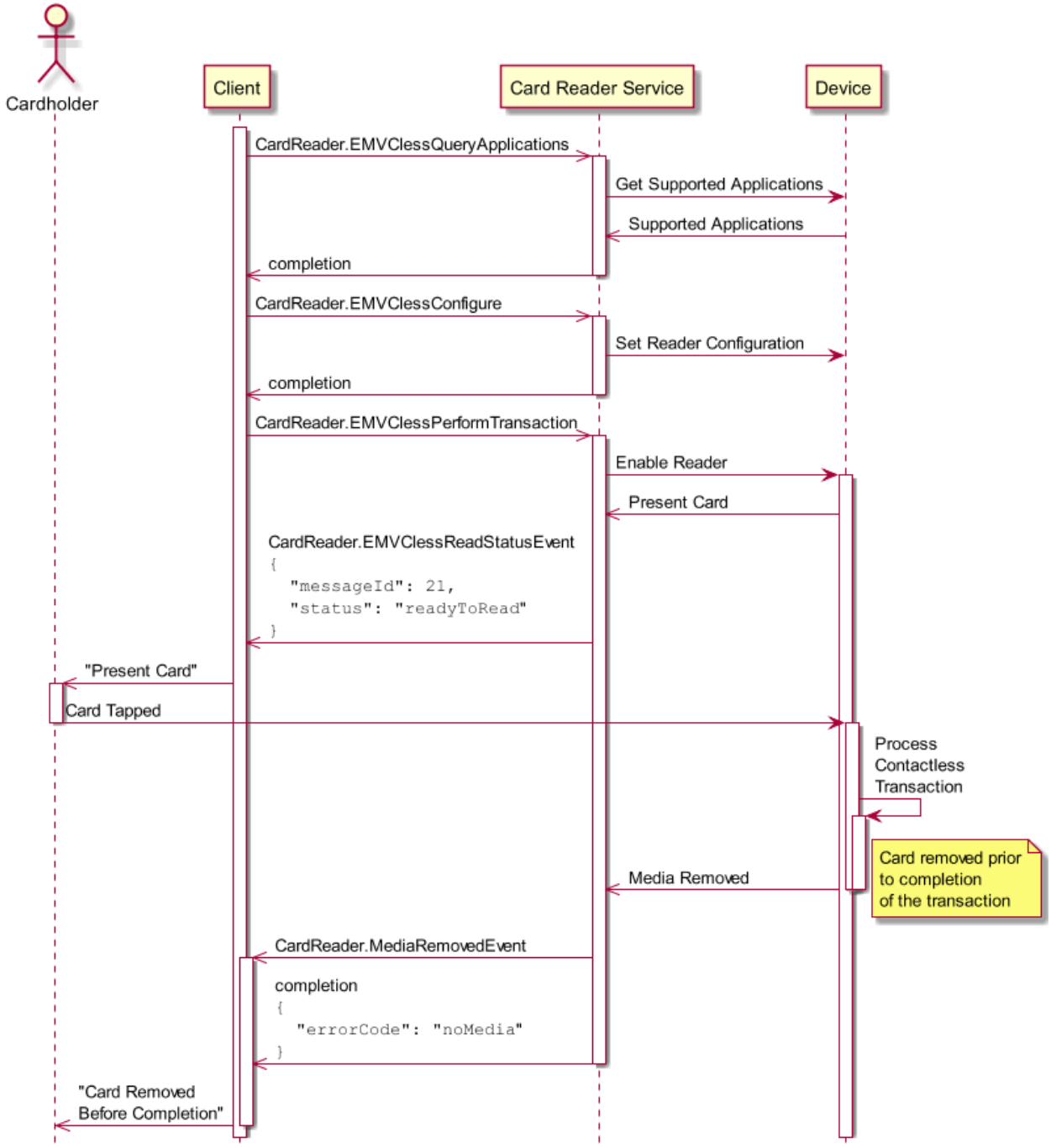
Single Tap Transaction Without Issuer Update Processing



Double Tap Transaction With Issuer Update Processing



Card Removed Before Completion



5.2 Command Messages

5.2.1 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 any additional properties included in *ifmIDs*.

Command Message

Payload (version 1.0)	Type	Required
`{`		
`"timeout": 5000`	integer	
`}`		

Properties

timeout

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

default: 0

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	string	
`"ifmIdentifiers": {`	object	
`"emv": "Example IFM Identifier",`	string	
`"europay": "Example IFM Identifier"`	string	
`}`		
`}`		

Properties

completionCode

The completion code.

errorDescription

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

ifmIdentifiers/emv (example name)

The object name representing IFM authority and contains IFM identifier of the chip card reader (or IFM) as assigned by the specified authority. The following IFM authorities are available:

- `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.

Property name constraints:

```
pattern: ^emv$|^europay$|^visa$|^giecb$
```

Event Messages

None

5.2.2 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 [Ref. cardreader-3].

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"appData": [{	array (object)	
"aid": "oAAAAAMQEA==",	string	
"kernelIdentifier": "Ag=="	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 application identifier (AID) and associated Kernel Identifier.

appData/aid
Contains the Base64 encoded payment system application identifier (AID) supported by the intelligent contactless card unit.
Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

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.
Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

Event Messages

None

5.2.3 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 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 this command 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, the *cardCollision* error will be returned.

Command Message

Payload (version 1.0)	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	
}		

Properties

timeout

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

default: 0

Properties
track1 Track 1 of the magnetic stripe will be read. default: false
track2 Track 2 of the magnetic stripe will be read. default: false
track3 Track 3 of the magnetic stripe will be read. default: false
chip The chip will be read. default: false
security A security check will be performed. default: false
fluxInactive If the Flux Sensor is programmable it will be disabled in order to allow chip data to be read on cards which have no magnetic stripes. default: false
watermark The Swedish Watermark track will be read. default: false
memoryChip The memory chip will be read. default: false
track1Front Track 1 data is read from the magnetic stripe located on the front of the card. In some countries this track is known as JIS II track. default: false
frontImage The front image of the card will be read in Base64 PNG format. default: false
backImage The back image of the card will be read in Base64 PNG format. default: false
track1JIS Track 1 of Japanese cash transfer card will be read. In some countries this track is known as JIS I track 1 (8bits/char). default: false
track3JIS Track 3 of Japanese cash transfer card will be read. In some countries this track is known as JIS I track 3 (8bits/char). default: false
ddi Dynamic Digital Identification data of the magnetic stripe will be read. default: false

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "mediaJam",	string	
"track1": {	object	
"status": "ok",	string	
"data": "QmFzZTY0IGVuY29kZWQg ..."	string	
},		
"track2": {	object	
See track1 properties.		
},		
"track3": {	object	
See track1 properties.		
},		
"chip": [{	array (object)	
See track1 properties.		
}],		
"security": {	object	
"status": "ok",	string	
"data": "readLevel1"	string	
},		
"watermark": {	object	
See track1 properties.		
},		
"memoryChip": {	object	
"status": "ok",	string	
"protocol": "chipT0",	string	
"data": "O2gAUACFyEARAJAC"	string	
},		
"track1Front": {	object	
See track1 properties.		
},		
"frontImage": "Add example",	string	
"backImage": "Add example",	string	
"track1JIS": {	object	
See track1 properties.		
},		
"track3JIS": {	object	
See track1 properties.		
},		

Payload (version 1.0)	Type	Required
"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.

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.
- `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

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

track2

Contains the data read from track 2.

Properties
track3 Contains the data read from track 3.
chip Contains the ATR data read from the chip. For contactless chip card readers, multiple identification information can be returned if the card reader detects more than one chip. Each chip identification information is returned as an individual *data* array element.
security Contains the data returned by the security module.
security/data The security data can be one of the following: • `readLevel1` - The security data readability level is 1. • `readLevel2` - The security data readability level is 2. • `readLevel3` - The security data readability level is 3. • `readLevel4` - The security data readability level is 4. • `readLevel5` - The security data readability level is 5. • `badReadLevel` - The security data reading quality is not acceptable. • `noData` - There are no security data on the card. • `dataInvalid` - The validation of the security data with the specific data on the magnetic stripe was not successful. • `hardwareError` - The security module could not be used because of a hardware error. • `notInitialized` - The security module could not be used because it was not initialized (e.g. CIM key is not loaded).
watermark Contains the data read from the Swedish Watermark track.
memoryChip Memory Card Identification data read from the memory chip.
memoryChip/protocol The memory card protocol used to communicate with the card. It can be one of the following: • `chipT0` - The card reader has used the T=0 protocol. • `chipT1` - The card reader has used the T=1 protocol. • `chipTypeAPart3` - The card reader has used the ISO 14443 (Part3) Type A contactless chip card protocol. • `chipTypeAPart4` - The card reader has used the ISO 14443 (Part4) Type A contactless chip card protocol. • `chipTypeB` - The card reader has used the ISO 14443 Type B contactless chip card protocol. • `chipTypeNFC` - The card reader has used the ISO 18092 (106/212/424kbps) contactless chip card protocol.
memoryChip/data Contains the data read from the memory chip in Base64.
track1Front Contains the data read from the front track 1. In some countries this track is known as JIS II track.

frontImage
Base64 encoded representation of the BMP image file for the front of the card.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

Properties

backImage
Base64 encoded representation of the BMP image file for the back of the card.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

track1JIS Contains the data read from JIS I track 1 (8bits/char).
track3JIS Contains the data read from JIS I track 3 (8bits/char).
ddi Contains the dynamic digital identification data read from magnetic stripe.

Event Messages

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

5.2.4 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, encoded in Base64 (See CardReader.ReadRawData). If the data passed in is too long the *invalidData* 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 (version 1.0)	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"data": [{`	array (object)	
`"destination": "track1",`	string	
`"data": "QmFzZTY0IGVuY29kZWQg ...",`	string	
`"writeMethod": "auto"`	string	
`}]`		
`}`		

Properties

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

data
An array of card data write instructions

data/
A card data write instruction

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
Base64 encoded representation of the data
Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

Properties
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 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 (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	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` - 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.InvalidMediaEvent

5.2.5 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 (version 1.0)	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"from": "unit1",`	string	
`"to": "exit"`	string	
`}`		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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$|^transport$|^unit[0-9A-Za-z]+$
```

default: "transport"

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$|^transport$|^unit[0-9A-Za-z]+$
```

default: "exit"

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	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.
- `full` - The *to* position is full. The card is still in the device.
- `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

5.2.6 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"keyValue": "QmFzZTY0IGVuY29kZWQg ..."	string	
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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.

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

Completion Message

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

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	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

5.2.7 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* or 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"chipProtocol": "chipT0",	string	
"chipData": "wCAAAQgwMDAwMDAwMA=="	string	
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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 chipProtocols. This property is ignored in communications with Memory Cards. The Service knows which memory card type is currently inserted and therefore there is no need for the application to manage this.

It 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 will automatically determine the protocol used to communicate with the chip.
- `chipTypeAPart3` - Use the ISO 14443 (Part3) Type A contactless chip card protocol to communicate with the chip.
- `chipTypeAPart4` - Use the ISO 14443 (Part4) Type A contactless chip card protocol to communicate with the chip.
- `chipTypeB` - Use the ISO 14443 Type B contactless chip card protocol to communicate with the chip.
- `chipTypeNFC` - Use the ISO 18092 (106/212/424kbps) contactless chip card protocol to communicate with the chip.

chipData

The Base64 encoded data to be sent to the chip.

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "mediaJam",	string	
"chipProtocol": "chipT0",	string	
"chipData": "bGs="	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.
- `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.
- `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.

It can be one of the following:

- `chipT0` - The T=0 protocol has been used to communicate with the chip.
- `chipT1` - The T=1 protocol has been used to communicate with the chip.
- `chipProtocolNotRequired` - The Service has automatically determined the protocol used to communicate with the chip.
- `chipTypeAPart3` - The ISO 14443 (Part3) Type A contactless chip card protocol has been used to communicate with the chip.
- `chipTypeAPart4` - The ISO 14443 (Part4) Type A contactless chip card protocol has been used to communicate with the chip.
- `chipTypeB` - The ISO 14443 Type B contactless chip card protocol has been used to communicate with the chip.
- `chipTypeNFC` - The ISO 18092 (106/212/424kbps) contactless chip card protocol has been used to communicate with the chip.

chipData

The Base64 encoded data received from the chip.

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

Event Messages

- CardReader.MediaRemovedEvent

5.2.8 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 properties, the device will attempt to move a card in transport or exit positions to the exit or transport positions or a retain storage unit.
- For each card in the device (including parking storage units), a CardReader.MediaDetectedEvent will indicate the position or state of the card on completion of this command.
- Dependent on device state, it may not be possible to move a card.

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

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"to": "retain",	string	
"storageId": "unit1"	string	
}		

Properties

timeout

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

default: 0

to

Specifies the position a card in the transport or exit position 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.
- `currentPosition` - Keep the card in its current position. If the card is in the transport, it may be moved in the transport to verify it is not jammed.

If omitted, the service will select the position to which the card will be moved based on device capabilities, retain storage units available and service specific configuration.

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 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: ^unit[0-9A-Za-z]+$
```

Completion Message

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

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "mediaJam"	string	

Payload (version 1.0)	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.
- `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

5.2.9 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"chipPower": "cold"	string	
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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 (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "chipPowerNotSupported",	string	
"chipData": "O2gAUACFyEARAJAC"	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:

- `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).

Properties

chipData
The Base64 encoded data received from the chip.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

Event Messages

- CardReader.MediaRemovedEvent

5.2.10 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. 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"terminalData": "Add example",	string	
"aidData": [{	array (object)	
"aid": "oAAAAAMQEA==",	string	
"partialSelection": false,	boolean	
"transactionType": 0,	integer	
"kernelIdentifier": "Ag==",	string	
"configData": "nwYHoAAAASFHEQ=="	string	
}],		
"keyData": [{	array (object)	
"rid": "oAAAAAM=",	string	
"caPublicKey": {	object	
"index": 0,	integer	
"algorithmIndicator": 0,	integer	
"exponent": "AQAB",	string	
"modulus": "Kjyq8qcAWnJB66p3cREs ...",	string	
"checksum": "7hURzscQIKm5BEOzex1f ..."	string	
}		
}]		
}		

Properties

timeout

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

default: 0

terminalData

Base64 encoded representation of the BER-TLV formatted data for the terminal e.g. Terminal Type, Transaction Category Code, Merchant Name & Location etc. Any terminal based data elements referenced in the Payment Systems Specifications or EMVCo Contactless Payment Systems Specifications Books may be included (see [Ref. cardreader-1], [Ref. cardreader-2] and [Ref. cardreader-3] for more details).

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

Properties
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 [Ref. cardreader-2] and [Ref. cardreader-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.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

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 [Ref. cardreader-2], 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 [Ref. cardreader-4].

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.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

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.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

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 [Ref. cardreader-2]. For example, if the EMV specification indicates the algorithm is ‘01’, the value of the algorithm is coded as 1.

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 [Ref. cardreader-2]. For example, representing value '$2^{16} + 1$' requires 3 bytes in hexadecimal (0x01, 0x00, 0x01), while value '3' is coded as 0x03.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

keyData/caPublicKey/modulus
Base64 encoded representation of the CA Public Key Modulus for the specific RID.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

keyData/caPublicKey/checksum
Base64 encoded representation of the 20 byte checksum value for the CA Public Key.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	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

5.2.11 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 (version 1.0)	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"data": "XyoCCXiaAxcICJwBAJ8C ..."`	string	
`}`		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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.

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	

Payload (version 1.0)	Type	Required
"errorDescription": "Device not available",	string	
"errorCode": "noMedia",	string	
"chip": {	object	
"txOutcome": "multipleCards",	string	
"cardholderAction": "none",	string	
"dataRead": "fSfILqum6niI6jURWzeo ...",	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": "fSfILqum6niI6jURWzeo ..."	string	
}		
},		
"track1": {	object	
See chip properties.		
},		
"track2": {	object	
See chip properties.		
},		
"track3": {	object	
See chip properties.		
}		
}		

Properties

completionCode

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

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:

- `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.

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.

chip/txOutcome

If multiple data sources are returned, this property is the same for each one. Specifies the contactless transaction outcome as one of the following:

- `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 Contactless Specifications for Payment Systems (Book A and B) [Ref. cardreader-3].*

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 *onlineRequest*, *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.

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

Properties
chip/clessOutcome The Entry Point Outcome specified in EMVCo Specifications for Contactless Payment Systems (Book A and B) [Ref. cardreader-3]. 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.
chip/clessOutcome/alternateInterface If txOutcome is not *tryAnotherInterface*, this is ignored and can be omitted. 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 [Ref. cardreader-3], 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 [Ref. cardreader-3], 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. Property value constraints: `minimum: 0` default: 0

Properties
chip/clessOutcome/uiOutcome/valueQualifier Qualifies *value*. This data is defined by EMVCo as one of the following. If neither apply, this field and *value* are omitted: • `amount` - *value* is an Amount. • `balance` - *value* is a Balance.
chip/clessOutcome/uiOutcome/value Represents the value of the amount or balance (as specified by *valueQualifier*) to be displayed where appropriate. If *valueQualifier* is omitted, this property is omitted.
chip/clessOutcome/uiOutcome/currencyCode Represents the numeric value of currency code as per ISO 4217. If omitted, the currency code is not available. Property value constraints: `pattern: ^[A-Z]{3}$`
chip/clessOutcome/uiOutcome/languagePreferenceData Represents the language preference (EMV Tag '5F2D') if returned by the card. If not returned, this property is omitted. The application should use this data to display all messages in the specified language until the transaction concludes. Property value constraints: `pattern: ^[a-z]{2}$`
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. Property value constraints: `minimum: 0`
chip/clessOutcome/cardRemovalTimeout Specifies a timeout value in units of 100 milliseconds for prompting the user to remove the card. Property value constraints: `minimum: 0` default: 0

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.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

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.
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.

Event Messages

- CardReader.EMVClessReadStatusEvent
- CardReader.MediaRemovedEvent

5.2.12 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 command request for a card to be tapped.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"data": "XyoCCXiaAxcICJwBAJ8C ..."	string	Yes
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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 *data* 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.

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "noMedia",	string	
"chip": {	object	
"txOutcome": "multipleCards",	string	
"dataRead": "fSfILqum6niI6jURWzeo ...",	string	
"clessOutcome": {	object	
"cvm": "onlinePIN",	string	
"alternateInterface": "contact",	string	
"receipt": false,	boolean	
"uiOutcome": {	object	

Payload (version 1.0)	Type	Required
"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": "fSfILqum6niI6jURWzeo ..."	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.

chip/txOutcome

If multiple data sources are returned, this property is the same for each one. Specifies the contactless transaction outcome as one of the following:

- `multipleCards` - Transaction could not be completed as more than one contactless card was tapped.
- `approve` - Transaction was approved offline.
- `decline` - Transaction was declined offline.
- `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).

Note: *The values for outcome have been mapped against the EMV Entry Point Outcome structure values defined in the EMVCo Contactless Specifications for Payment Systems (Book A and B) [Ref. cardreader-3].*

Properties

chip/dataRead

The Base64 encoded representation of the data read from the chip after a contactless transaction has been completed successfully. The BER-TLV formatted data contains cryptogram tag (9F26) after a contactless chip transaction has been completed successfully. This value is terminated with a single null character and cannot contain UNICODE characters.

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

chip/clessOutcome

The Entry Point Outcome specified in EMVCo Specifications for Contactless Payment Systems (Book A and B) [Ref. cardreader-3]. 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.

chip/clessOutcome/alternateInterface

If txOutcome is not *tryAnotherInterface*, this ignored and can be omitted. 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 [Ref. cardreader-3], 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 [Ref. cardreader-3], 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.

Properties
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. Property value constraints: `minimum: 0` default: 0
chip/clessOutcome/uiOutcome/valueQualifier Qualifies *value*. This data is defined by EMVCo as one of the following. If neither apply, this field and *value* are omitted: • `amount` - *value* is an Amount. • `balance` - *value* is a Balance.
chip/clessOutcome/uiOutcome/value Represents the value of the amount or balance (as specified by *valueQualifier*) to be displayed where appropriate. If *valueQualifier* is omitted, this property is omitted.
chip/clessOutcome/uiOutcome/currencyCode Represents the numeric value of currency code as per ISO 4217. If omitted, the currency code is not available. Property value constraints: `pattern: ^[A-Z]{3}$`
chip/clessOutcome/uiOutcome/languagePreferenceData Represents the language preference (EMV Tag '5F2D') if returned by the card. If not returned, this property is omitted. The application should use this data to display all messages in the specified language until the transaction concludes. Property value constraints: `pattern: ^[a-z]{2}$`
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. Property value constraints: `minimum: 0`
chip/clessOutcome/cardRemovalTimeout Specifies a timeout value in units of 100 milliseconds for prompting the user to remove the card. Property value constraints: `minimum: 0` default: 0

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.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

Event Messages

- CardReader.EMVClessReadStatusEvent

- CardReader.MediaRemovedEvent

5.3 Event Messages

5.3.1 CardReader.InsertCardEvent

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

Event Message

Payload (version 1.0)	Type	Required
{		
}		

5.3.2 CardReader.MediaInsertedEvent

This event specifies that a card was inserted into the device.

Event Message

Payload (version 1.0)	Type	Required
{		
}		

5.3.3 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.

Event Message

Payload (version 1.0)	Type	Required
{		
}		

5.3.4 CardReader.TrackDetectedEvent

This 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.

Event Message

Payload (version 1.0)	Type	Required
{		
"track1": false,	boolean	
"track2": false,	boolean	
"track3": false,	boolean	
"watermark": false,	boolean	
"frontTrack1": false	boolean	
}		

Properties
track1 The card has track 1. default: false
track2 The card has track 2. default: false
track3 The card has track 3. default: false
watermark The card has the Swedish watermark track. default: false
frontTrack1 The card has front track 1. default: false

5.3.5 CardReader.MediaDetectedEvent

This is generated if media is detected during a CardReader.Reset. The event payload informs the application of the position or state of a card on the completion of the *CardReader.Reset* command. For devices with park storage units, there will be one event for each card found.

Event Message

Payload (version 1.0)	Type	Required
`{`		
`"position": "unit1"`	string	
`}`		

Properties

position

Specifies a card position or jammed state as one of the following:

- `exit` - A card is at the exit position.
- `transport` - A card is in the transport position.
- `<storage unit identifier>` - A card is in the identified *retain* or *park* storage unit.
- `jammed` - A card is jammed in the device.

Property value constraints:

```
pattern: ^exit$|^transport$|^jammed$|^unit[0-9A-Za-z]+$
```

5.3.6 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.

Event Message

Payload (version 1.0)	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 [Ref. cardreader-3], 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.

Property value constraints:

```
minimum: 0
```

default: 0

valueQualifier

Qualifies *value*. This data is defined by EMVCo as one of the following. If neither apply, this field and *value* are omitted:

- `amount` - *value* is an Amount.
- `balance` - *value* is a Balance.

value

Represents the value of the amount or balance (as specified by *valueQualifier*) to be displayed where appropriate. If *valueQualifier* is omitted, this property is omitted.

Properties
currencyCode Represents the numeric value of currency code as per ISO 4217. If omitted, the currency code is not available. Property value constraints: `pattern: ^[A-Z]{3}$`
languagePreferenceData Represents the language preference (EMV Tag ‘5F2D’) if returned by the card. If not returned, this property is omitted. The application should use this data to display all messages in the specified language until the transaction concludes. Property value constraints: `pattern: ^[a-z]{2}$`

5.4 Unsolicited Messages

5.4.1 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.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
}		

5.4.2 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.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
"to": "unit1",	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$|^transport$|^unit[0-9A-Za-z]+$
```

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 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$|^unit[0-9A-Za-z]+$
```

6. Cash Management Interface

This chapter defines the Cash Management interface functionality and messages.

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 Storage, Cash Dispenser and/or Cash Acceptor interfaces to handle management of storage units, cash counts and banknote information.

6.1 General Information

6.1.1 References

ID	Description
cashmanagement-1	ISO 4217

6.1.2 Note Classification

Cash items are classified by the XFS4IoT specification according to the following definitions. Local requirements or device capability define which of these classifications are supported. A cash item can only be classified as one of the following:

1. Not recognized (level 1 in XFS 3.x), defined as *unrecognized* in XFS4IoT.
2. Recognized counterfeit item (level 2 in XFS 3.x), defined as *counterfeit* in XFS4IoT.
3. Suspected counterfeit item (level 3 in XFS 3.x), defined as *suspect* in XFS4IoT.
4. Inked, defined as *inked* in XFS4IoT. Inked-stained banknotes are typically items which have been stained by anti-theft devices.
5. Genuine note (level 4 in XFS 3.x). Genuine items are further classified as follows:

- Fit for recycling, defined as *fit* in XFS4IoT
- Unfit for recycling, defined as *unfit* in XFS4IoT

Once classified as such, how items are handled may depend on local requirements or legislative note handling standards that may exist in various countries and economic regions. This can be used to support note handling functionality which includes:

1. Whether counterfeit or suspect items allowed to be returned to the customer during a Cash In transaction
2. The ability to remove counterfeit notes from circulation.
3. Reporting of recognized, counterfeit and suspected counterfeit notes.
4. Creating and reporting of note signatures in order to allow back-tracing of notes.

A note's classification can be changed based on the item's serial number, currency and value by specifying a classification list - see CashManagement.SetClassificationList. A classification list can be used to re-classify a matching item 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 note retention rules. Any reclassification will result in the normal events and behavior, for example a CashManagement.InfoAvailableEvent 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 (commonly known as a blacklist).

Reclassification cannot be used to change a note's classification to a level which makes it more likely to be accepted, 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 counterfeit note as unrecognized. No particular use case has been identified for reclassifying suspect or genuine items as unrecognized, but there is no reason to restrict this reclassification.

Classification lists can be specified using CashManagement.SetClassificationList and retrieved using CashManagement.GetClassificationList.

The classification list functionality can use a mask to specify serial numbers. The mask is defined as follows: A '?' character (0x003F) is the wildcard used to match a single Unicode character, and a '*' character (0x002A) is the wildcard used to match one or more Unicode characters.

For example, "S8H9??16?4" would represent a match for the serial numbers "S8H9231654" and "S8H9761684". A mask of "HD90*2" would be used in order to match serial numbers that begin with "HD90" and end with "2", for example "HD9028882", "HD9083276112". Note that the mask can only use one asterisk, and if a real character is required then it must be preceded by a backslash, for example: '\\' for a backslash, '*' for an asterisk or '\?' for a question mark. Note that this flexibility means that it is possible to overlap definitions, for example "HD90*" and "HD902*" would both match on the serial number HD9028882".

6.2 Command Messages

6.2.1 CashManagement.GetBankNoteTypes

This command is used to obtain information about the banknote types that can be detected by the banknote reader or are supported by the configuration.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"items": {	object	
"*type20USD1*": {	object	
"cashItem": {	object	
"noteID": 25,	integer	
"currency": "USD",	string	
"value": 20,	number	
"release": 1	integer	
},		
"enabled": true	boolean	
},		
"*type10GBP2*": {	object	
See type20USD1 properties.		
}		
}		
}		

Properties

completionCode
The completion code.

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

Properties
items An object listing which cash items the device is capable of handling and whether the cash items are enabled for acceptance.
items/type20USD1 (example name) Specifies a cash item supported by the device and whether it is enabled for acceptance. Property name constraints: `pattern: ^type[0-9A-Z]+$`
items/type20USD1/cashItem An object containing information about a single cash item supported by the device.
items/type20USD1/cashItem/noteID Assigned by the Service. A unique number identifying a single cash item. Each unique combination of the other properties will have a different noteID. Can be used for migration of *usNoteID* from XFS 3.x. Property value constraints: `minimum: 1`
items/type20USD1/cashItem/currency ISO 4217 currency identifier [Ref. cashmanagement-1]. Property value constraints: `pattern: ^[A-Z]{3}$`
items/type20USD1/cashItem/value Absolute value of a cash item or items. 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. If applied to a storage unit, this applies to all contents, may be 0 if mixed and may only be modified in an exchange state if applicable. Property value constraints: `minimum: 0`
items/type20USD1/cashItem/release The release of the cash item. The higher this number is, the newer the release. If 0 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 description configuration data 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`
items/type20USD1/enabled If true the banknote reader will accept this note type during a cash-in operations. If false the banknote reader will refuse this note type unless it must be retained by note classification rules. default: true

Event Messages

None

6.2.2 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. The teller information is persistent.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"tellerID": 0,	integer	
"currency": "USD"	string	
}		

Properties

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

tellerID
Identification of the teller. If invalid the error *invalidTellerId* is reported. If not specified, all tellers are reported.
Property value constraints:

```
minimum: 0
```

currency
ISO 4217 format currency identifier [Ref. cashmanagement-1]. If not specified, all currencies are reported for *tellerID*.
Property value constraints:

```
pattern: ^[A-Z]{3}$
```

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "invalidCurrency",	string	
"tellerDetails": [{	array (object)	
"tellerID": 104,	integer	
"inputPosition": "inDefault",	string	
"outputPosition": "outDefault",	string	
"tellerTotals": {	object	
"*EUR*": {	object	
"itemsReceived": 1407.15,	number	
"itemsDispensed": 0,	number	
"coinsReceived": 0.05,	number	
"coinsDispensed": 0,	number	

Payload (version 1.0)	Type	Required
"cashBoxReceived": 0,	number	
"cashBoxDispensed": 0	number	
},		
"*GBP": {*	object	
See EUR 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. 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.

Property value constraints:

```
minimum: 0
```

tellerDetails/inputPosition

Supplies the input position as one of the following values. Supported positions are reported in Common.Capabilities.

- `inDefault` - Default input position.
- `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.

default: "inDefault"

Properties
tellerDetails/outputPosition Supplies the output position as one of the following values. Supported positions are reported in Common.Capabilities. • `outDefault` - Default output 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. default: "outDefault"
tellerDetails/tellerTotals List of teller total objects. There is one object per currency.
tellerDetails/tellerTotals/EUR (example name) The property name is the ISO 4217 currency identifier [Ref. cashmanagement-1]. Property name constraints: `pattern: ^[A-Z]{3}$`
tellerDetails/tellerTotals/EUR/itemsReceived The total absolute value of items (other than coins) of the specified currency accepted. The amount is expressed as a floating point value. Property value constraints: `minimum: 0`
tellerDetails/tellerTotals/EUR/itemsDispensed The total absolute value of items (other than coins) of the specified currency dispensed. The amount is expressed as a floating point value. Property value constraints: `minimum: 0`
tellerDetails/tellerTotals/EUR/coinsReceived The total absolute value of coin currency accepted. The amount is expressed as a floating point value. Property value constraints: `minimum: 0`
tellerDetails/tellerTotals/EUR/coinsDispensed The total absolute value of coin currency dispensed. The amount is expressed as a floating point value. Property value constraints: `minimum: 0`
tellerDetails/tellerTotals/EUR/cashBoxReceived The total absolute value of cash box currency accepted. The amount is expressed as a floating point value. Property value constraints: `minimum: 0`
tellerDetails/tellerTotals/EUR/cashBoxDispensed The total absolute value of cash box currency dispensed. The amount is expressed as a floating point value. Property value constraints: `minimum: 0`

Event Messages

None

6.2.3 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"action": "create",	string	
"tellerDetails": {	object	
"tellerID": 104,	integer	
"inputPosition": "inDefault",	string	
"outputPosition": "outDefault",	string	
"tellerTotals": {	object	
"*EUR*": {	object	
"itemsReceived": 1407.15,	number	
"itemsDispensed": 0,	number	
"coinsReceived": 0.05,	number	
"coinsDispensed": 0,	number	
"cashBoxReceived": 0,	number	
"cashBoxDispensed": 0	number	
},		
"*GBP": {*	object	
See EUR properties.		
}		
}		
}		
}		

Properties

timeout

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

default: 0

action

The action to be performed. Following values are possible:

- `create` - A teller is to be added.
- `modify` - Information about an existing teller is to be modified.
- `delete` - A teller is to be removed.

tellerDetails

Teller details object.

tellerDetails/tellerID

Identification of the teller.

Property value constraints:

```
minimum: 0
```

Properties
tellerDetails/inputPosition Supplies the input position as one of the following values. Supported positions are reported in Common.Capabilities. • `inDefault` - Default input position. • `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. default: "inDefault"
tellerDetails/outputPosition Supplies the output position as one of the following values. Supported positions are reported in Common.Capabilities. • `outDefault` - Default output 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. default: "outDefault"
tellerDetails/tellerTotals List of teller total objects. There is one object per currency.
tellerDetails/tellerTotals/EUR (example name) The property name is the ISO 4217 currency identifier [Ref. cashmanagement-1]. Property name constraints: `pattern: ^[A-Z]{3}$`
tellerDetails/tellerTotals/EUR/itemsReceived The total absolute value of items (other than coins) of the specified currency accepted. The amount is expressed as a floating point value. Property value constraints: `minimum: 0`
tellerDetails/tellerTotals/EUR/itemsDispensed The total absolute value of items (other than coins) of the specified currency dispensed. The amount is expressed as a floating point value. Property value constraints: `minimum: 0`
tellerDetails/tellerTotals/EUR/coinsReceived The total absolute value of coin currency accepted. The amount is expressed as a floating point value. Property value constraints: `minimum: 0`
tellerDetails/tellerTotals/EUR/coinsDispensed The total absolute value of coin currency dispensed. The amount is expressed as a floating point value. Property value constraints: `minimum: 0`

Properties
tellerDetails/tellerTotals/EUR/cashBoxReceived The total absolute value of cash box currency accepted. The amount is expressed as a floating point value. Property value constraints: `minimum: 0`
tellerDetails/tellerTotals/EUR/cashBoxDispensed The total absolute value of cash box currency dispensed. The amount is expressed as a floating point value. Property value constraints: `minimum: 0`

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	string	
`"errorCode": "invalidCurrency"`	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. 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

6.2.4 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 classification, or all items that have information available. This information is available from the point where the first CashManagement.InfoAvailableEvent is generated until a transaction or replenishment command is executed including the following:

- CashAcceptor.CashInStart
- CashAcceptor.CashIn
- CashAcceptor.CashInEnd
- CashAcceptor.CashInRollback
- CashAcceptor.CreateSignature
- CashAcceptor.Replenish
- CashAcceptor.CashUnitCount
- CashAcceptor.Deplete
- CashManagement.Retract
- CashManagement.Reset
- CashManagement.OpenShutter
- CashManagement.CloseShutter
- CashManagement.CalibrateCashUnit
- CashDispenser.Dispense
- CashDispenser.Present
- CashDispenser.Reject
- CashDispenser.Count
- CashDispenser.TestCashUnits
- Storage.StartExchange
- Storage.EndExchange

In addition, since the item information is not cumulative and can be replaced by any command that can move notes, it is recommended that applications that are interested in the available information should query for it following the *CashManagement.InfoAvailableEvent* but before any other command is executed.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"items": {	object	
"level": "fit",	string	
"index": 1	integer	
},		
"itemInfoType": {	object	
"serialNumber": false,	boolean	
"signature": false,	boolean	
"image": false	boolean	
}		
}		

Properties

timeout

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

default: 0

Properties
items Specifies which item or items to return information for. If not specified, all information on all items is returned.
items/level Specifies the item's classification. Following values are possible: • `unrecognized` - The item is not recognized. • `counterfeit` - The item is recognized as counterfeit. • `suspect` - The item is recognized as suspected counterfeit. • `fit` - The item is genuine and fit for recycling. • `unfit` - The item is genuine but not fit for recycling. • `inked` - The item is genuine but ink stained.
items/index Specifies the zero based index for the item information required. If not specified, all items of the specified *level* will be returned. Property value constraints: `minimum: 0`
itemInfoType Specifies the type of information required. If not specified, all available information will be returned.
itemInfoType/serialNumber Request the serial number of the item.
itemInfoType/signature Request the signature of the item.
itemInfoType/image Request the image of the item.

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	string	
`"itemsList": [{`	array (object)	
`"noteType": "type20USD1",`	string	
`"orientation": "frontTop",`	string	
`"signature": "MAA5ADgANwA2ADUANAAz ...",`	string	
`"level": "fit",`	string	
`"serialNumber": "AB12345YG",`	string	
`"image": "MAA5ADgANwA2ADUANAAz ...",`	string	
`"onClassificationList": "onClassificationList",`	string	
`"itemLocation": "unit1"`	string	
`}]`		
`}`		

Properties
completionCode The completion code.

Properties

errorDescription

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

itemsList

Array of objects listing the item information.

itemsList/noteType

A cash item as reported by CashManagement.GetBankNoteTypes. This is not specified if the item was not identified as a cash item.

Property value constraints:

```
pattern: ^type[0-9A-Z]+$
```

itemsList/orientation

Specifies the note orientation. The following values are possible:

- `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.
- `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.
- `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.
- `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.
- `unknown` - The orientation for the inserted note cannot be determined.
- `notSupported` - The hardware is not capable to determine the orientation.

itemsList/signature

Base64 encoded vendor specific signature data. If no signature is available or has not been requested then this is omitted.

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

itemsList/level

Specifies the item's classification. Following values are possible:

- `unrecognized` - The item is not recognized.
- `counterfeit` - The item is recognized as counterfeit.
- `suspect` - The item is recognized as suspected counterfeit.
- `fit` - The item is genuine and fit for recycling.
- `unfit` - The item is genuine but not fit for recycling.
- `inked` - The item is genuine but ink stained.

itemsList/serialNumber

This property contains the serial number of the item as a string. A '?' character is used to represent any serial number character that cannot be recognized. If no serial number is available or has not been requested then this is omitted.

itemsList/image

Base64 encoded binary image data. If the Service does not support this function or the image has not been requested then this is omitted.

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

Properties
itemsList/onClassificationList Specifies if the item is on the classification list. If the classification list reporting capability is not supported this property 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: • `customer` - The item has been presented to the customer. • `unknown` - The item location is unknown, for example, it may have been removed manually. • `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 in an intermediate location in the device. • `deviceUnknown` - The item is in the device but its location is unknown. • `<storage unit identifier>` - The item is in a storage unit with matching identifier. Property value constraints: `pattern: ^customer$\|^unknown$|^stacker$\|^output$|^transport$\|^deviceUnknown$|^unit[0-9A-Za-z]+$`

Event Messages

None

6.2.5 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 (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"version": "Version 1.2",	string	
"classificationElements": [{	array (object)	
"serialNumber": "AB1234D",	string	
"currency": "USD",	string	
"value": 20,	number	
"level": "fit"	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 objects defining the classification list.

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 Note Classification.

Properties
classificationElements/currency ISO 4217 currency identifier [Ref. cashmanagement-1]. Property value constraints: `pattern: ^[A-Z]{3}$`
classificationElements/value Absolute value of a cash item or items. 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. If applied to a storage unit, this applies to all contents, may be 0 if mixed and may only be modified in an exchange state if applicable. Property value constraints: `minimum: 0`
classificationElements/level Specifies the item's classification. Following values are possible: • `unrecognized` - The item is not recognized. • `counterfeit` - The item is recognized as counterfeit. • `suspect` - The item is recognized as suspected counterfeit. • `fit` - The item is genuine and fit for recycling. • `unfit` - The item is genuine but not fit for recycling. • `inked` - The item is genuine but ink stained.

Event Messages

None

6.2.6 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 counterfeit note as unrecognized. If two or more classification elements specify overlapping note definitions, but different *level* values then the first one takes priority.

Command Message

Payload (version 1.0)	Type	Required

{
  "timeout": 5000,
  "version": "Version 1.2",
  "classificationElements": [{
  "serialNumber": "AB1234D",
  "currency": "USD",
  "value": 20,
  "level": "fit"
  }]
}
	integer
	string
	array (object)
	string
	string
	number
	string	

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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 objects defining the classification list.

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 Note Classification.

classificationElements/currency

ISO 4217 currency identifier [Ref. cashmanagement-1].

Property value constraints:

```
pattern: ^[A-Z]{3}$
```

classificationElements/value

Absolute value of a cash item or items. 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.

If applied to a storage unit, this applies to all contents, may be 0 if mixed and may only be modified in an exchange state if applicable.

Property value constraints:

```
minimum: 0
```

Properties
classificationElements/level Specifies the item's classification. Following values are possible: • `unrecognized` - The item is not recognized. • `counterfeit` - The item is recognized as counterfeit. • `suspect` - The item is recognized as suspected counterfeit. • `fit` - The item is genuine and fit for recycling. • `unfit` - The item is genuine but not fit for recycling. • `inked` - The item is genuine but ink stained.

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available"`	string	
`}`		

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

Event Messages

None

6.2.7 CashManagement.CloseShutter

This command closes the shutter.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"position": "inLeft"	string	
}		

Properties

timeout

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

default: 0

position

Supplies the input or output position as one of the following values. If not specified, the default position applies. Supported positions are reported in Common.Capabilities.

- `inDefault` - Default input position.
- `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.
- `outDefault` - Default output 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.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "unsupportedPosition"	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.

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. • `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

None

6.2.8 CashManagement.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. 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"position": "inLeft"	string	
}		

Properties

timeout

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

default: 0

position

Supplies the input or output position as one of the following values. If not specified, the default position applies. Supported positions are reported in Common.Capabilities.

- `inDefault` - Default input position.
- `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.
- `outDefault` - Default output 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.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "unsupportedPosition"	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. 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. • `foreignItemsDetected` - Foreign items have been detected in the input position.

Event Messages

- CashManagement.ItemsTakenEvent
- CashManagement.ItemsInsertedEvent
- CashManagement.ItemsPresentedEvent

6.2.9 CashManagement.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 storage 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 shutterControl is false.

This command terminates a running cash-in transaction. The cash-in transaction is terminated even if this command does not complete successfully.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"outputPosition": "outDefault",	string	
"retractArea": "retract",	string	
"index": 0	integer	
}		

Properties

timeout

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

default: 0

outputPosition

Supplies the output position as one of the following values. Supported positions are reported in Common.Capabilities.

- `outDefault` - Default output 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.

default: "outDefault"

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 storage unit.
- `transport` - Retract the items to the transport.
- `stacker` - Retract the items to the intermediate stacker area.
- `reject` - Retract the items to a reject storage unit.
- `itemCassette` - Retract the items to the storage units which would be used during a Cash In transaction including recycling storage units.
- `cashIn` - Retract the items to the storage units which would be used during a Cash In transaction but not including recycling storage units.

Properties
index If *retractArea* is set to *retract* this property defines the position inside the retract storage units into which the cash is to be retracted. *index* starts with a value of 1 for the first retract position and increments by one for each subsequent position. If there are several retract storage units (of type *retractCassette* in Storage.GetStorage), *index* would be incremented from the first position of the first retract storage unit to the last position of the last retract storage unit. The maximum value of *index* is the sum of *maximum* of each retract storage unit. If *retractArea* is not set to *retract* the value of this property is ignored.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "cashUnitError",	string	
"storage": {	object	
"*unit1*": {	object	
"retractOperations": 0,	integer	
"deposited": {	object	
"unrecognized": 0,	integer	
"*type20USD1*": {	object	
"fit": 0,	integer	
"unfit": 0,	integer	
"suspect": 0,	integer	
"counterfeit": 0,	integer	
"inked": 0	integer	
},		
"*type50USD1": {*	object	
See type20USD1 properties.		
}		
},		
"retracted": {	object	
See deposited properties.		
},		
"rejected": {	object	
See deposited properties.		
},		
"distributed": {	object	
See deposited properties.		
},		
"transport": {	object	
See deposited properties.		
}		

Payload (version 1.0)	Type	Required
},		
"*unit2*": {	object	
See unit1 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. Following values are possible:

- `cashUnitError` - A problem occurred with a storage unit. A Storage.StorageErrorEvent 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.
- `incompleteRetract` - Some or all of the items were not retracted for a reason not covered by other error codes. The detail will be reported with the Dispenser.IncompleteRetractEvent.

storage

List of storage units that have taken items and the type of items they have taken during the current command.

storage/unit1 (example name)

List of items moved to this storage unit by this transaction or command. The property name is the same as reported by Storage.GetStorage.

Property name constraints:

```
pattern: ^unit[0-9A-Za-z]+$
```

storage/unit1/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/deposited

The items deposited in the storage unit during a Cash In transaction.

storage/unit1/deposited/unrecognized

Count of unrecognized items handled by the cash interface.

storage/unit1/deposited/type20USD1 (example name)

Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification.

storage/unit1/deposited/type20USD1/fit

Count of genuine cash items which are fit for recycling.

storage/unit1/deposited/type20USD1/unfit

Count of genuine cash items which are unfit for recycling.

Properties
storage/unit1/deposited/type20USD1/suspect Count of suspected counterfeit cash items.
storage/unit1/deposited/type20USD1/counterfeit Count of counterfeit cash items.
storage/unit1/deposited/type20USD1/inked Count of cash items which have been identified as ink stained.
storage/unit1/retracted The items retracted into 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/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/distributed The items deposited in this storage unit originating from another storage unit but not rejected.
storage/unit1/transport The items which were intended to be deposited in this storage unit but are not yet deposited. Typical use case for this property is tracking items after a jam during CashAcceptor.CashInEnd. This is not reset if initial is set for this unit by Storage.GetStorage.

Event Messages

- Storage.StorageChangedEvent
- Storage.StorageErrorEvent
- Storage.StorageThresholdEvent
- CashManagement.NoteErrorEvent
- CashManagement.ItemsTakenEvent
- CashManagement.InfoAvailableEvent
- CashManagement.IncompleteRetractEvent

6.2.10 CashManagement.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 override a lock obtained on another application or connection.

If a cash-in transaction is active or exchange is *active* then this command will end the transaction or exchange state as appropriate, 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 command parameters. This may not always be possible because of hardware problems. If the application does not wish to specify a storage unit or position it can leave the command payload empty. In this case the Device will determine where to move any items found.

If items are found inside the device one or more CashManagement.MediaDetectedEvents will be generated to inform the application where the items have actually been moved to.

The shutterControl property will determine whether the shutter is controlled implicitly by this command or whether the application must explicitly open and close the shutter using the CashManagement.OpenShutter, CashManagement.CloseShutter or CashAcceptor.PresentMedia commands. 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 property 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 *CashManagement.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.

Command Message

Payload (version 1.0)	Type	Required

{
  "timeout": 5000,
  "unit": "unit1",
  "retractArea": {
  "outputPosition": "outDefault",
  "retractArea": "retract",
  "index": 0
  },
  "outputPosition": "outDefault"
}
	object	

Properties

timeout

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

default: 0

unit

Specifies the object name (as stated by the Storage.GetStorage command) of the single unit to be used for the storage of any items found.

Property value constraints:

```
pattern: ^unit[0-9A-Za-z]+$
```

Properties
retractArea This property is used if items are to be moved to internal areas of the device, including storage units, the intermediate stacker, or the transport.
retractArea/outputPosition Supplies the output position as one of the following values. Supported positions are reported in Common.Capabilities. • `outDefault` - Default output 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. default: "outDefault"
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 storage unit. • `transport` - Retract the items to the transport. • `stacker` - Retract the items to the intermediate stacker area. • `reject` - Retract the items to a reject storage unit. • `itemCassette` - Retract the items to the storage units which would be used during a Cash In transaction including recycling storage units. • `cashIn` - Retract the items to the storage units which would be used during a Cash In transaction but not including recycling storage units.
retractArea/index If *retractArea* is set to *retract* this property defines the position inside the retract storage units into which the cash is to be retracted. *index* starts with a value of 1 for the first retract position and increments by one for each subsequent position. If there are several retract storage units (of type *retractCassette* in Storage.GetStorage), *index* would be incremented from the first position of the first retract storage unit to the last position of the last retract storage unit. The maximum value of *index* is the sum of *maximum* of each retract storage unit. If *retractArea* is not set to *retract* the value of this property is ignored.

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	string	
`"errorCode": "cashUnitError"`	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.

Properties
errorCode Specifies the error code if applicable. Following values are possible: • `cashUnitError` - There is a problem with a storage unit. A Storage.StorageErrorEvent will be posted with the details. • `unsupportedPosition` - The output position specified is not supported. • `invalidCashUnit` - The storage 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. • `foreignItemsDetected` - Foreign items have been detected in the input position. • `incompleteRetract` - Some or all of the items were not retracted for a reason not covered by other error codes. The detail will be reported with the Dispenser.IncompleteRetractEvent.

Event Messages

- Storage.StorageErrorEvent
- Storage.StorageThresholdEvent
- CashManagement.MediaDetectedEvent
- CashManagement.ItemsTakenEvent
- CashManagement.InfoAvailableEvent

6.2.11 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 (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "exchangeActive"	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. Following values are possible:

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

Event Messages

None

6.2.12 CashManagement.CalibrateCashUnit

This command will cause a vendor dependent sequence of hardware events which will calibrate one storage unit. This is necessary if a new type of bank note is put into the storage unit as the command enables the device to obtain the measures of the new bank notes.

This command cannot be used to calibrate storage units which have been locked by the application. An error code will be returned and a Storage.StorageErrorEvent generated.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"unit": "unit1",	string	
"numOfBills": 40,	integer	
"position": {	object	
"unit": "unit1",	string	
"retractArea": {	object	
"outputPosition": "outDefault",	string	
"retractArea": "retract",	string	
"index": 0	integer	
},		
"outputPosition": "outDefault"	string	
}		
}		

Properties

timeout

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

default: 0

unit

The object name of the storage unit as stated by the Storage.GetStorage command.

Property value constraints:

```
pattern: ^unit[0-9A-Za-z]+$
```

numOfBills

The number of bills to be dispensed during the calibration process. If not specified or 0, the Service may decide how many bills are required. This may also be ignored if the device always dispenses a default number of bills.

Property value constraints:

```
minimum: 0
```

position

Defines where items are to be moved or have been moved as one of the following:

- A single storage unit, specified by *unit*.
- Internal areas of the device, specified by *retractArea*.
- Output position, specified by *outputPosition*.

position/unit

Specifies the object name (as stated by the Storage.GetStorage command) of the single unit to be used for the storage of any items found.

Property value constraints:

```
pattern: ^unit[0-9A-Za-z]+$
```

Properties
position/retractArea This property is used if items are to be moved to internal areas of the device, including storage units, the intermediate stacker, or the transport.
position/retractArea/outputPosition Supplies the output position as one of the following values. Supported positions are reported in Common.Capabilities. • `outDefault` - Default output 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. default: "outDefault"
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 storage unit. • `transport` - Retract the items to the transport. • `stacker` - Retract the items to the intermediate stacker area. • `reject` - Retract the items to a reject storage unit. • `itemCassette` - Retract the items to the storage units which would be used during a Cash In transaction including recycling storage units. • `cashIn` - Retract the items to the storage units which would be used during a Cash In transaction but not including recycling storage units.
position/retractArea/index If *retractArea* is set to *retract* this property defines the position inside the retract storage units into which the cash is to be retracted. *index* starts with a value of 1 for the first retract position and increments by one for each subsequent position. If there are several retract storage units (of type *retractCassette* in Storage.GetStorage), *index* would be incremented from the first position of the first retract storage unit to the last position of the last retract storage unit. The maximum value of *index* is the sum of *maximum* of each retract storage unit. If *retractArea* is not set to *retract* the value of this property is ignored.

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	string	
`"errorCode": "cashUnitError",`	string	
`"unit": "unit1",`	string	
`"numOfBills": 20,`	integer	
`"position": {`	object	
`"unit": "unit1",`	string	
`"retractArea": {`	object	
`"outputPosition": "outDefault",`	string	
`"retractArea": "retract",`	string	
`"index": 0`	integer	
`},`		

Payload (version 1.0)	Type	Required
"outputPosition": "outDefault"	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. Following values are possible:

- `cashUnitError` - A storage unit caused an error. A Storage.StorageErrorEvent will be sent with the details.
- `unsupportedPosition` - The position specified is not valid.
- `exchangeActive` - The device is in an exchange state.
- `invalidCashUnit` - The storage unit number specified is not valid.

unit

The object name of the storage unit which has been calibrated as stated by Storage.GetStorage.

Property value constraints:

```
pattern: ^unit[0-9A-Za-z]+$
```

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 device 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 storage unit counts.

Property value constraints:

```
minimum: 0
```

position

Specifies where the items were moved to during the calibration process.

position/unit

Specifies the object name (as stated by the Storage.GetStorage command) of the single unit to be used for the storage of any items found.

Property value constraints:

```
pattern: ^unit[0-9A-Za-z]+$
```

position/retractArea

This property is used if items are to be moved to internal areas of the device, including storage units, the intermediate stacker, or the transport.

Properties
position/retractArea/outputPosition Supplies the output position as one of the following values. Supported positions are reported in Common.Capabilities. • `outDefault` - Default output 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. default: "outDefault"
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 storage unit. • `transport` - Retract the items to the transport. • `stacker` - Retract the items to the intermediate stacker area. • `reject` - Retract the items to a reject storage unit. • `itemCassette` - Retract the items to the storage units which would be used during a Cash In transaction including recycling storage units. • `cashIn` - Retract the items to the storage units which would be used during a Cash In transaction but not including recycling storage units.
position/retractArea/index If *retractArea* is set to *retract* this property defines the position inside the retract storage units into which the cash is to be retracted. *index* starts with a value of 1 for the first retract position and increments by one for each subsequent position. If there are several retract storage units (of type *retractCassette* in Storage.GetStorage), *index* would be incremented from the first position of the first retract storage unit to the last position of the last retract storage unit. The maximum value of *index* is the sum of *maximum* of each retract storage unit. If *retractArea* is not set to *retract* the value of this property is ignored.

Event Messages

- Storage.StorageChangedEvent
- Storage.StorageErrorEvent
- Storage.StorageThresholdEvent
- CashManagement.NoteErrorEvent
- CashManagement.InfoAvailableEvent
- CashManagement.ItemsTakenEvent

6.3 Event Messages

6.3.1 CashManagement.TellerInfoChangedEvent

This 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.

Event Message

Payload (version 1.0)	Type	Required
{		
"tellerID": 0	integer	
}		

Properties

tellerID

Integer holding the ID of the teller whose counts have changed.

Property value constraints:

```
minimum: 0
```

6.3.2 CashManagement.NoteErrorEvent

This event specifies the reason for a note detection error during the execution of a command.

Event Message

Payload (version 1.0)	Type	Required
{		
"reason": "doubleNote"	string	
}		

Properties

reason

The reason for the notes detection error. Following values are possible:

- `doubleNote` - A double note has 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` - A short note has been detected.

6.3.3 CashManagement.InfoAvailableEvent

This event is generated when information is available for items detected during the cash processing operation.

Event Message

Payload (version 1.0)	Type	Required
{		
"itemInfoSummary": [{	array (object)	
"level": "fit",	string	
"numOfItems": 2	integer	
}]		
}		

Properties

itemInfoSummary
Array of itemInfoSummary objects, one object for every level.

itemInfoSummary/level
Specifies the item's classification. Following values are possible:

- `unrecognized` - The item is not recognized.
- `counterfeit` - The item is recognized as counterfeit.
- `suspect` - The item is recognized as suspected counterfeit.
- `fit` - The item is genuine and fit for recycling.
- `unfit` - The item is genuine but not fit for recycling.
- `inked` - The item is genuine but ink stained.

itemInfoSummary/numOfItems
Number of items classified as *level* which have information available.
Property value constraints:

```
minimum: 1
```

6.3.4 CashManagement.IncompleteRetractEvent

This event is generated when an attempt to retract items has completed with an error and not all of the items have been retracted.

Event Message

Payload (version 1.0)	Type	Required
{		
"itemNumberList": {	object	
"*unit1*": {	object	
"retractOperations": 0,	integer	
"deposited": {	object	
"unrecognized": 0,	integer	
"*type20USD1*": {	object	
"fit": 0,	integer	
"unfit": 0,	integer	
"suspect": 0,	integer	
"counterfeit": 0,	integer	
"inked": 0	integer	
},		
"*type50USD1*": {	object	
See type20USD1 properties.		
}		
},		
"retracted": {	object	
See deposited properties.		
},		
"rejected": {	object	
See deposited properties.		
},		
"distributed": {	object	
See deposited properties.		
},		
"transport": {	object	
See deposited properties.		
}		
},		
"*unit2*": {	object	
See unit1 properties.		
}		
},		
"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/unit1 (example name) List of items moved to this storage unit by this transaction or command. The property name is the same as reported by Storage.GetStorage. Property name constraints: `pattern: ^unit[0-9A-Za-z]+$`
itemNumberList/unit1/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.
itemNumberList/unit1/deposited The items deposited in the storage unit during a Cash In transaction.
itemNumberList/unit1/deposited/unrecognized Count of unrecognized items handled by the cash interface.
itemNumberList/unit1/deposited/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification.
itemNumberList/unit1/deposited/type20USD1/fit Count of genuine cash items which are fit for recycling.
itemNumberList/unit1/deposited/type20USD1/unfit Count of genuine cash items which are unfit for recycling.
itemNumberList/unit1/deposited/type20USD1/suspect Count of suspected counterfeit cash items.
itemNumberList/unit1/deposited/type20USD1/counterfeit Count of counterfeit cash items.
itemNumberList/unit1/deposited/type20USD1/inked Count of cash items which have been identified as ink stained.
itemNumberList/unit1/retracted The items retracted into 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*.
itemNumberList/unit1/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.
itemNumberList/unit1/distributed The items deposited in this storage unit originating from another storage unit but not rejected.
itemNumberList/unit1/transport The items which were intended to be deposited in this storage unit but are not yet deposited. Typical use case for this property is tracking items after a jam during CashAcceptor.CashInEnd. This is not reset if initial is set for this unit by Storage.GetStorage.

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 storage area specified in the command payload 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.

6.3.5 CashManagement.MediaDetectedEvent

This is generated if media is detected during a CashManagement.Reset command. The payload specifies the position of the media on completion of the command. If the device has been unable to successfully move the items found then this parameter will be omitted.

Event Message

Payload (version 1.0)	Type	Required
{		
"unit": "unit1",	string	
"retractArea": {	object	
"outputPosition": "outDefault",	string	
"retractArea": "retract",	string	
"index": 0	integer	
},		
"outputPosition": "outDefault"	string	
}		

Properties

Defines where items are to be moved or have been moved as one of the following:

- A single storage unit, specified by *unit*.
- Internal areas of the device, specified by *retractArea*.
- Output position, specified by *outputPosition*.

unit

Specifies the object name (as stated by the Storage.GetStorage command) of the single unit to be used for the storage of any items found.

Property value constraints:

```
pattern: ^unit[0-9A-Za-z]+$
```

retractArea

This property is used if items are to be moved to internal areas of the device, including storage units, the intermediate stacker, or the transport.

retractArea/outputPosition

Supplies the output position as one of the following values. Supported positions are reported in Common.Capabilities.

- `outDefault` - Default output 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.

default: "outDefault"

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 storage unit. • `transport` - Retract the items to the transport. • `stacker` - Retract the items to the intermediate stacker area. • `reject` - Retract the items to a reject storage unit. • `itemCassette` - Retract the items to the storage units which would be used during a Cash In transaction including recycling storage units. • `cashIn` - Retract the items to the storage units which would be used during a Cash In transaction but not including recycling storage units.
retractArea/index If *retractArea* is set to *retract* this property defines the position inside the retract storage units into which the cash is to be retracted. *index* starts with a value of 1 for the first retract position and increments by one for each subsequent position. If there are several retract storage units (of type *retractCassette* in Storage.GetStorage), *index* would be incremented from the first position of the first retract storage unit to the last position of the last retract storage unit. The maximum value of *index* is the sum of *maximum* of each retract storage unit. If *retractArea* is not set to *retract* the value of this property is ignored.

6.4 Unsolicited Messages

6.4.1 CashManagement.SafeDoorOpenEvent

This event specifies that the safe door has been opened.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
}		

6.4.2 CashManagement.SafeDoorClosedEvent

This event specifies that the safe door has been closed.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
}		

6.4.3 CashManagement.ItemsTakenEvent

This specifies that items presented to the user have been taken. This event may be generated at any time

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
"position": {	object	
"position": "inLeft",	string	
"additionalBunches": "1"	string	
}		
}		

Properties

position/position

Supplies the input or output position as one of the following values. If not specified, the default position applies. Supported positions are reported in Common.Capabilities.

- `inDefault` - Default input position.
- `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.
- `outDefault` - Default output 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.

position/additionalBunches

Specifies how many more bunches will be required to present the request. Following values are possible:

- `<number>` - The number of additional bunches to be presented.
- `unknown` - More than one additional bunch is required but the precise number is unknown.

Property value constraints:

```
pattern: ^unknown$|^[0-9]*$
```

default: "0"

6.4.4 CashManagement.ItemsInsertedEvent

This specifies that items have been inserted into the position by the user. This event may be generated at any time.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
"position": "inLeft"	string	
}		

Properties

position

Supplies the input or output position as one of the following values. If not specified, the default position applies. Supported positions are reported in Common.Capabilities.

- `inDefault` - Default input position.
- `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.
- `outDefault` - Default output 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.

6.4.5 CashManagement.ItemsPresentedEvent

This specifies that items have been presented to the user, and the shutter has been opened to allow the user to take the items.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
"position": {	object	
"position": "inLeft",	string	
"additionalBunches": "1"	string	
}		
}		

Properties

position/position

Supplies the input or output position as one of the following values. If not specified, the default position applies. Supported positions are reported in Common.Capabilities.

- `inDefault` - Default input position.
- `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.
- `outDefault` - Default output 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.

position/additionalBunches

Specifies how many more bunches will be required to present the request. Following values are possible:

- `<number>` - The number of additional bunches to be presented.
- `unknown` - More than one additional bunch is required but the precise number is unknown.

Property value constraints:

```
pattern: ^unknown$|^[0-9]*$
```

default: "0"

6.4.6 CashManagement.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.

Unsolicited Message

Payload (version 1.0)	Type	Required
`{`		
`"position": "inLeft",`	string	
`"shutter": "closed"`	string	
`}`		

Properties

position

Supplies the input or output position as one of the following values. If not specified, the default position applies. Supported positions are reported in Common.Capabilities.

- `inDefault` - Default input position.
- `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.
- `outDefault` - Default output 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 state of the shutter. Following values are possible:

- `closed` - The shutter is fully 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.

7. Cash Dispenser Interface

This chapter defines the Cash Dispenser interface functionality and messages.

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 sessions 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.

7.1 General Information

7.1.1 References

ID	Description
cashdispenser-1	ISO 4217

7.2 Command Messages

7.2.1 CashDispenser.GetMixTypes

This command is used to obtain a list of supported mix algorithms and available house mix tables.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"mixes": {	object	
"*mix1*": {	object	
"type": "algorithm",	string	
"algorithm": "minimumBills",	string	
"name": "Minimum Bills"	string	
},		
"*mixIndividual*": {	object	
See mix1 properties.		
}		
}		
}		

Properties

completionCode
The completion code.

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

mixes
Object containing mix specifications including mix tables and pre-defined algorithms. The property name of each mix can be used as the *mix* in the CashDispenser.Dispense and CashDispenser.Denominate commands.
Mix tables are defined by CashDispenser.SetMixTable. A mix table's definition can be queried using its property name as input to CashDispenser.GetMixTable.

Properties
mixes/mix1 (example name) An object containing a single mix specification. The property name is assigned by the Service. Property name constraints: `pattern: ^mix[0-9A-Za-z]+$`
mixes/mix1/type Specifies the mix type as one of the following: • `individual` - the mix is not calculated by the Service, completely specified by the application. • `algorithm` - the mix is calculated using one of the algorithms specified by *algorithm*. • `table` - the mix is calculated using a mix table - see CashDispenser.GetMixTable.
mixes/mix1/algorithm If *type* is *algorithm*, specifies the algorithm type as one of the following. There are three pre-defined algorithms, additional vendor-defined algorithms can also be defined. Omitted if the mix is not an algorithm. • `minimumBills` - Select a mix requiring the minimum possible number of items. • `equalEmptying` - The denomination is selected based upon criteria which ensure that over the course of its operation the storage units will empty as far as possible at the same rate and will therefore go low and then empty at approximately the same time. • `maxCashUnits` - The denomination is selected based upon criteria which ensures the maximum number of storage units are used. • `<vendor-defined mix>` - A vendor defined mix algorithm. Property value constraints: `pattern: ^minimumBills$\|^equalEmptying$|^maxCashUnits$\|^[A-Za-z0-9]*$`
mixes/mix1/name Name of the table or algorithm used. May be omitted.

Event Messages

None

7.2.2 CashDispenser.GetMixTable

This command is used to obtain the specified house mix table. Mix tables can be set using CashDispenser.SetMixTable.

Command Message

Payload (version 1.0)	Type	Required

{
"timeout": 5000,
"mix": "mixTable21"
}
	integer
	string	

Properties

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

mix
A house mix table as defined by one of the mixes reported by CashDispenser.GetMixTypes.

Completion Message

Payload (version 1.0)	Type	Required

{
"completionCode": "success",
"errorDescription": "Device not available",
"errorCode": "invalidMix",
"mixNumber": 21,
"name": "House mix 21",
"mixRows": [{
"amount": 0.30,
"mix": [{
"value": 0.05,
"count": 6
}]
}]
}
	array (object)
	number
	array (object)
	number
	integer	

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:

- `invalidMix` - The *mix* property does not correspond to a defined mix table.

Properties
mixNumber Number identifying the house mix table. Property value constraints: `minimum: 1`
name Name of the house mix table.
mixRows Array of rows of the mix table.
mixRows/amount Absolute value of the amount denominated by this mix row. Property value constraints: `minimum: 0`
mixRows/mix The items used to create *amount*. Each element in this array defines the quantity of a given item used to create the mix. An example showing how 0.30 can be broken down would be: `[ { "value": 0.05, "count": 2 }, { "value": 0.10, "count": 2 } ]`
mixRows/mix/ The quantity of a given absolute value of cash items contained in the mix.
mixRows/mix/value The absolute value of a single cash item. Property value constraints: `minimum: 0`
mixRows/mix/count The number of items of *value* contained in the mix. Property value constraints: `minimum: 1`

Event Messages

None

7.2.3 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.

The denominations reported by this command may not accurately reflect the operation if the storage units have been re-configured, e.g., if the values associated with a storage unit are changed, or new storage units are configured.

If end-to-end security is supported then this value is *not* cleared if a *CashDispenser.Dispense* with an invalid token is received. If a dispense token is invalid the dispense will fail with an *invalidToken* error, and the 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.

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 this command 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.

Command Message

Payload (version 1.0)	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"position": "outDefault",`	string	
`"nonce": "646169ECDD0E440C2CECC8DDD7C27C22"`	string	
`}`		

Properties

timeout

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

default: 0

position

Supplies the output position as one of the following values. Supported positions are reported in Common.Capabilities.

- `outDefault` - Default output 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.

default: "outDefault"

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 (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "unsupportedPosition",	string	
"denomination": {	object	
"currencies": {	object	
"*EUR*": 10,	number	
"*USD*": *10*	number	
},		
"values": {	object	
"*unit1*": 5,	integer	
"*unit2*": *5*	integer	
},		
"cashBox": {	object	
See currencies properties.		
}		
},		
"presentState": "presented",	string	
"token": "NONCE=1414,TOKENFORMAT=1,TOKENLENGTH ..."	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. 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 storage unit. This is cumulative across a series of CashDispenser.Dispense calls that add additional items to the stacker.
denomination/currencies List of currency and amount combinations for denomination requests or output. There will be one entry for each currency in the denomination. This list can be omitted on a request if *values* specifies the entire request.

Properties

denomination/currencies/EUR (example name)
The absolute amount to be or which has been denominated or dispensed of the currency. The property name is the ISO 4217 currency identifier [Ref. cashdispenser-1]. The property value can include a decimal point to specify fractions of the currency, for example coins.
Property name constraints:
`pattern: ^[A-Z]{3}$`
Property value constraints:
`minimum: 0.001`

denomination/values This list specifies the number of items to take or which have been taken from the storage units. If specified in a request, the output denomination must include these items. The property name is storage unit object name as stated by the Storage.GetStorage command. The value of the entry is the number of items to take from that unit.

denomination/values/unit1 (example name)
The number of items have been dispensed from the specified storage unit to meet the request.
Property name constraints:
`pattern: ^unit[0-9A-Za-z]+$`
Property value constraints:
`minimum: 1`

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 end-to-end security for more information. An example is `NONCE=1414,TOKENFORMAT=1,TOKENLENGTH=0268,DISPENSEID=CB735612FD6141213C2827FB5A6A4F4846D7A7347B15434916FEA6AC16F3D2F2,DISPENSED1=50.00EUR,PRESENTED1=YES,PRESENTEDAMOUNT1=50.00EUR,RETRACTED1=NO,HMACSHA256=55D123E9EE64F0CC3D1CD4F953348B441E521BBACCD6998C6F51D645D71E6C83`

Event Messages

None

7.2.4 CashDispenser.Denominate

This command provides a denomination which specifies the number of items which are required from each storage unit in order to satisfy a given request and can be used to validate that any request supplied by the application can be dispensed.

The request may contain the following items:

- The amount of each currency to be included in the denomination (*currencies*).
- The number of items which must be included from individual storage units (*counts*).
- If the Service is to calculate a denomination, the method (algorithm or table) by which the denomination is to be calculated (*mix*).

If the request contains *currencies*, the denomination is calculated as follows:

- If *counts* are specified, these are validated against *currencies* for consistency.
- If the total amount specified by *counts* is greater than the amount specified by *currencies*, an error is returned.
- If the total amount specified by *counts* is less than specified by *currencies*, this is defined as a *partial denomination* where a calculated denomination must contain at least the items specified by *counts*. The additional items needed to create the denomination are calculated using *mix*, taking the number of items and availability of each storage unit into account.
- If the total amount specified by *counts* is equal to the amount specified by *currencies*, the number of items and availability of each requested storage unit is taken into account.

If the request does not contain *currencies*, *counts* is validated against the number of items and availability of each requested storage unit.

If cashBox is true, then if the entire request cannot be satisfied, the denomination will include an amount to be supplied from the teller's cash box.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"denomination": {	object	
"currencies": {	object	
"*EUR*": 10,	number	
"*USD": 10*	number	
},		
"values": {	object	
"*unit1*": 5,	integer	
"*unit2": 5*	integer	
},		
"cashBox": {	object	
See currencies properties.		
}		
},		
"mix": "mix1",	string	
"tellerID": 0	integer	
}		

Properties
timeout Timeout in milliseconds for the command to complete. If set to 0, the command will not timeout but can be canceled. default: 0
denomination Specifies a denomination or a denomination request.
denomination/currencies List of currency and amount combinations for denomination requests or output. There will be one entry for each currency in the denomination. This list can be omitted on a request if *values* specifies the entire request.

denomination/currencies/EUR (example name)
The absolute amount to be or which has been denominated or dispensed of the currency. The property name is the ISO 4217 currency identifier [Ref. cashdispenser-1]. The property value can include a decimal point to specify fractions of the currency, for example coins.
Property name constraints:
`pattern: ^[A-Z]{3}$`
Property value constraints:
`minimum: 0.001`

denomination/values This list specifies the number of items to take or which have been taken from the storage units. If specified in a request, the output denomination must include these items. The property name is storage unit object name as stated by the Storage.GetStorage command. The value of the entry is the number of items to take from that unit.

denomination/values/unit1 (example name)
The number of items have been dispensed from the specified storage unit to meet the request.
Property name constraints:
`pattern: ^unit[0-9A-Za-z]+$`
Property value constraints:
`minimum: 1`

denomination/cashBox Only applies to Teller Dispensers. Amount to be paid from the teller's cash box.
mix Mix algorithm or house mix table to be used as defined by mixes reported by CashDispenser.GetMixTypes. May be omitted if the request is entirely specified by *counts*. Property value constraints: `pattern: ^mix[0-9A-Za-z]+$`
tellerID Only applies to Teller Dispensers. Identification of teller.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "invalidCurrency",	string	
"currencies": {	object	
"*EUR*": 10,	number	
"*USD*": *10*	number	

Payload (version 1.0)	Type	Required
},		
"values": {	object	
"unit1": 5,	integer	
"unit2": 5	integer	
},		
"cashBox": {	object	
See currencies 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. Following values are possible:

- `invalidCurrency` - There are no storage units in the device of the currency specified in the request.
- `invalidTellerID` - Invalid teller ID. This error will never be generated by a Self-Service device.
- `cashUnitError` - There is a problem with a storage unit. A Storage.StorageErrorEvent will be posted with the details.
- `invalidDenomination` - No *mix* is specified and the sum of the values for *counts* and
- cashBox* does not match the non-zero *currencies* specified.
- `invalidMixNumber` - Unknown mix algorithm.
- `noCurrencyMix` - The storage units specified in the request 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 a unit is specified in the *counts* list which is not a dispensing storage unit, e.g., a retract/reject storage unit.
- `tooManyItems` - The request requires too many items to be dispensed.
- `exchangeActive` - The device is in an exchange state (see Storage.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 in the request is not in the mix table.

currencies

List of currency and amount combinations for denomination requests or output. There will be one entry for each currency in the denomination. This list can be omitted on a request if *values* specifies the entire request.

currencies/EUR (example name)

The absolute amount to be or which has been denominated or dispensed of the currency. The property name is the ISO 4217 currency identifier [Ref. cashdispenser-1]. The property value can include a decimal point to specify fractions of the currency, for example coins.

Property name constraints:

```
pattern: ^[A-Z]{3}$
```

Property value constraints:

```
minimum: 0.001
```

Properties
values This list specifies the number of items to take or which have been taken from the storage units. If specified in a request, the output denomination must include these items. The property name is storage unit object name as stated by the Storage.GetStorage command. The value of the entry is the number of items to take from that unit.

values/unit1 (example name)
The number of items have been dispensed from the specified storage unit to meet the request.
Property name constraints:
`pattern: ^unit[0-9A-Za-z]+$`
Property value constraints:
`minimum: 1`

cashBox Only applies to Teller Dispensers. Amount to be paid from the teller's cash box.

Event Messages

- Storage.StorageErrorEvent

7.2.5 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* must be *mixIndividual*. However, these restrictions do not apply if a single currency is dispensed with non-currency items, such as coupons.

The command can be used in the following ways:

- The input parameters to the command are amounts, currencies and denomination. The *mix* is *mixIndividual*. In this case, the denomination is checked for validity and, if valid, is dispensed.
- The input parameters are amounts, currencies and a mix algorithm or table. In this case the amount is denominated and, if this succeeds, the items are dispensed.
- If the amount and currency information is omitted and a denomination is supplied with a *mix* of *mixIndividual*, the denomination is checked for validity and, if valid, is dispensed.
- 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 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 cashBox is true and 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.

The CashDispenser.Present command is used 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 genuine 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 storage 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 *bunches* field in the completion message. If it is set to "unknown" or a value larger than "1" multiple presents will be necessary. If the value is set to "1" or omitted the dispensed amount can be presented in one present operation.

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.

It is possible to do multiple dispense and present operations in a row using the same dispense token, as long as the total value of cash doesn't exceed the value authorized by the token.

The device will track the command nonce and E2E token used during dispense operations. Only one token can be used with the current nonce - once a dispense command is called with a token then that token will be remembered, and it will not be possible to perform a dispense command with a different token until the original nonce and token are cleared.

The device will track the total value of cash that has been dispensed and presented using the current token. The device will block any attempt to dispense or present more cash than authorized by the current token.

Once the value of cash that has been dispensed and presented reaches the value of the token, the command nonce stored in the device will be cleared. This has the effect of making any existing tokens invalid so that they can't be used again. No more cash can be dispensed until a new command nonce is read and a new token is generated.

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.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"denomination": {	object	
"denomination": {	object	
"currencies": {	object	
"*EUR*": 10,	number	
"*USD": 10*	number	
},		
"values": {	object	
"*unit1*": 5,	integer	
"*unit2": 5*	integer	
},		
"cashBox": {	object	
See currencies properties.		
}		
},		
"mix": "mix1",	string	
"tellerID": 0	integer	
},		
"position": "outDefault",	string	
"token": "NONCE=254611E63B2531576314E86527338D61, ..."	string	
}		

Properties

timeout

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

default: 0

denomination

Denomination object describing the contents of the denomination operation.

denomination/denomination

Specifies a denomination or a denomination request.

denomination/denomination/currencies

List of currency and amount combinations for denomination requests or output. There will be one entry for each currency in the denomination. This list can be omitted on a request if *values* specifies the entire request.

denomination/denomination/currencies/EUR (example name)

The absolute amount to be or which has been denominated or dispensed of the currency. The property name is the ISO 4217 currency identifier [Ref. cashdispenser-1]. The property value can include a decimal point to specify fractions of the currency, for example coins.

Property name constraints:

```
pattern: ^[A-Z]{3}$
```

Property value constraints:

```
minimum: 0.001
```

Properties
denomination/denomination/values This list specifies the number of items to take or which have been taken from the storage units. If specified in a request, the output denomination must include these items. The property name is storage unit object name as stated by the Storage.GetStorage command. The value of the entry is the number of items to take from that unit.

denomination/denomination/values/unit1 (example name)
The number of items have been dispensed from the specified storage unit to meet the request.
Property name constraints:
`pattern: ^unit[0-9A-Za-z]+$`
Property value constraints:
`minimum: 1`

denomination/denomination/cashBox Only applies to Teller Dispensers. Amount to be paid from the teller's cash box.
denomination/mix Mix algorithm or house mix table to be used as defined by mixes reported by CashDispenser.GetMixTypes. May be omitted if the request is entirely specified by *counts*. Property value constraints: `pattern: ^mix[0-9A-Za-z]+$`
denomination/tellerID Only applies to Teller Dispensers. Identification of teller.
position If specified, defines the output position to which the items are to be dispensed. If not specified, the items are dispensed to one of the following positions as applicable: • teller position if the device is a Teller Dispenser • intermediate stacker if the device has one • the default position if there is no intermediate stacker. default: "outDefault"

Properties
token The dispense token that authorizes the dispense operation, as created by the authorizing host. See the section on end-to-end security for more information. The same token may be used multiple times with multiple calls to the CashDispenser.Dispense and CashDispenser.Present commands, as long as the total value stacked does not exceed the value given in the token. The hardware will track the total value of the cash and will raise an *invalidToken* error for any attempt to dispense or present more cash than authorized by the token. The token contains a nonce returned by Common.GetCommandNonce which must match the nonce stored in the hardware. The nonce value stored in the hardware will be cleared automatically at various times, meaning that all tokens will become invalid. The hardware will also track the token being used and block any attempt to use multiple tokens with the same nonce. The same token must be used for all calls to dispense, until the nonce is cleared and a new nonce and token is created. Any attempt to use a different token will trigger an *invalidToken* error. For maximum security the client should also explicitly clear the command nonce (and hence invalidate and existing tokens,) with the Common.ClearCommandNonce command as soon as it's finished using the current token. The dispense token will follow the standard token format, and will contain the standard keys plus the following key: `DISPENSE1`: The maximum value to be dispensed. This will be a number string that may contain a fractional part. The decimal character will be ".". The value, including the fractional part, will be defined by the ISO 4217 currency identifier [Ref. cashdispenser-1]. The number will be followed by the ISO 4217 currency code. The currency code will be upper case. For example, "123.45EUR" will be €123 and 45 cents. The "DISPENSE" key may appear multiple times with a number suffix. For example, DISPENSE1, DISPENSE2, DISPENSE3. The number will start at 1 and increment. Each key can only be given once. Each key must have a value in a different currency. For example, DISPENSE1=100.00EUR,DISPENSE2=200.00USD The actual amount dispensed will be given by the denomination. The value in the token MUST be greater or equal to the amount in the denomination parameter. If the Token has a lower value, or the Token is invalid for any reason, then the command will fail with an invalid data error code. Example token is as follows: `NONCE=254611E63B2531576314E86527338D61,TOKENFORMAT=1,TOKENLENGTH=0164,DISPENSE1=50.00EUR,HMACSHA256=CB735612FD6141213C2827FB5A6A4F4846D7A7347B15434916FEA6AC16F3D2F2`

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "invalidCurrency",	string	
"denomination": {	object	
"currencies": {	object	
"*EUR*": 10,	number	
"*USD*": 10	number	
},		
"values": {	object	
"*unit1*": 5,	integer	
"*unit2*": 5	integer	
},		
"cashBox": {	object	

Payload (version 1.0)	Type	Required
See currencies properties.		
}		
},		
"bunches": "1",	string	
"storage": {	object	
"in": {	object	
"*unit1*": {	object	
"retractOperations": 0,	integer	
"deposited": {	object	
"unrecognized": 0,	integer	
"*type20USD1*": {	object	
"fit": 0,	integer	
"unfit": 0,	integer	
"suspect": 0,	integer	
"counterfeit": 0,	integer	
"inked": 0	integer	
},		
"*type50USD1": {*	object	
See type20USD1 properties.		
}		
},		
"retracted": {	object	
See deposited properties.		
},		
"rejected": {	object	
See deposited properties.		
},		
"distributed": {	object	
See deposited properties.		
},		
"transport": {	object	
See deposited properties.		
}		
},		
"*unit2": {*	object	
See unit1 properties.		
}		
},		
"out": {	object	
"*unit3*": {	object	
"presented": {	object	

Payload (version 1.0)	Type	Required
See deposited properties.		
},		
"rejected": {	object	
See deposited properties.		
},		
"distributed": {	object	
See deposited properties.		
},		
"unknown": {	object	
See deposited properties.		
},		
"stacked": {	object	
See deposited properties.		
},		
"diverted": {	object	
See deposited properties.		
},		
"transport": {	object	
See deposited properties.		
}		
},		
"unit4": {	object	
See unit3 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.

Properties

errorCode

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

- `invalidCurrency` - There are no storage units in the device of the currency specified in the request.
- `invalidTellerID` - Invalid teller ID. This error will never be generated by a Self-Service device.
- `cashUnitError` - There is a problem with a storage unit. A Storage.StorageErrorEvent will be posted with the details.
- `invalidDenomination` - No *mix* is specified and the sum of the values for *counts* and
- cashBox* does not match the non-zero *currencies* specified.
- `invalidMixNumber` - Unknown mix algorithm.
- `noCurrencyMix` - The storage units specified in the request 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 a unit is specified in the *counts* list which is not a dispensing storage unit, e.g., a retract/reject storage unit.
- `tooManyItems` - The request requires too many items to be dispensed.
- `exchangeActive` - The device is in an exchange state (see Storage.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 in the request is not in the mix table.
- `unsupportedPosition` - The specified output position is not supported.
- `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.

denomination

Denomination object describing the contents of the denomination operation.

denomination/currencies

List of currency and amount combinations for denomination requests or output. There will be one entry for each currency in the denomination. This list can be omitted on a request if *values* specifies the entire request.

denomination/currencies/EUR (example name)

The absolute amount to be or which has been denominated or dispensed of the currency. The property name is the ISO 4217 currency identifier [Ref. cashdispenser-1]. The property value can include a decimal point to specify fractions of the currency, for example coins.

Property name constraints:

```
pattern: ^[A-Z]{3}$
```

Property value constraints:

```
minimum: 0.001
```

denomination/values

This list specifies the number of items to take or which have been taken from the storage units. If specified in a request, the output denomination must include these items.

The property name is storage unit object name as stated by the Storage.GetStorage command. The value of the entry is the number of items to take from that unit.

denomination/values/unit1 (example name)

The number of items have been dispensed from the specified storage unit to meet the request.

Property name constraints:

```
pattern: ^unit[0-9A-Za-z]+$
```

Property value constraints:

```
minimum: 1
```

denomination/cashBox

Only applies to Teller Dispensers. Amount to be paid from the teller's cash box.

Properties
bunches Specifies how many bunches will be required to present the request. Following values are possible: • `<number>` - The number of bunches to be presented. • `unknown` - More than one bunch is required but the precise number is unknown. Property value constraints: `pattern: ^unknown$\|^[0-9]*$` default: "1"
storage Object which lists the storage units which have had items removed or inserted during the associated operation or transaction. Only storage units whose contents have been modified are included.
storage/in Object containing the storage units which have had items inserted during the associated operation or transaction. Only storage units whose contents have been modified are included.
storage/in/unit1 (example name) List of items moved to this storage unit by this transaction or command. The property name is the same as reported by Storage.GetStorage. Property name constraints: `pattern: ^unit[0-9A-Za-z]+$`
storage/in/unit1/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/in/unit1/deposited The items deposited in the storage unit during a Cash In transaction.
storage/in/unit1/deposited/unrecognized Count of unrecognized items handled by the cash interface.
storage/in/unit1/deposited/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification.
storage/in/unit1/deposited/type20USD1/fit Count of genuine cash items which are fit for recycling.
storage/in/unit1/deposited/type20USD1/unfit Count of genuine cash items which are unfit for recycling.
storage/in/unit1/deposited/type20USD1/suspect Count of suspected counterfeit cash items.
storage/in/unit1/deposited/type20USD1/counterfeit Count of counterfeit cash items.
storage/in/unit1/deposited/type20USD1/inked Count of cash items which have been identified as ink stained.
storage/in/unit1/retracted The items retracted into 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/in/unit1/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/in/unit1/distributed The items deposited in this storage unit originating from another storage unit but not rejected.

Properties
storage/in/unit1/transport The items which were intended to be deposited in this storage unit but are not yet deposited. Typical use case for this property is tracking items after a jam during CashAcceptor.CashInEnd. This is not reset if initial is set for this unit by Storage.GetStorage.
storage/out Object containing the storage units which have had items removed during the associated operation or transaction. Only storage units whose contents have been modified are included.
storage/out/unit3 (example name) List of items removed from this storage unit by this transaction or command. The property name is the same as reported by Storage.GetStorage. Property name constraints: `pattern: ^unit[0-9A-Za-z]+$`
storage/out/unit3/presented The items dispensed from this storage unit which are or were customer accessible.
storage/out/unit3/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/out/unit3/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/out/unit3/unknown The items dispensed from this storage unit which moved to an unknown position.
storage/out/unit3/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. This is not reset if initial is set for this unit by Storage.GetStorage.
storage/out/unit3/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. This is not reset if initial is set for this unit by Storage.GetStorage.
storage/out/unit3/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. This is not reset if initial is set for this unit by Storage.GetStorage.

Event Messages

- Storage.StorageThresholdEvent
- CashDispenser.DelayedDispenseEvent
- CashDispenser.StartDispenseEvent
- Storage.StorageErrorEvent
- CashManagement.ItemsTakenEvent
- CashDispenser.IncompleteDispenseEvent
- CashManagement.NoteErrorEvent
- CashManagement.InfoAvailableEvent

7.2.6 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 shutterControl is false. The shutter will be closed when the user removes the items or the items are retracted. If the default *position* is specified 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 the previous *CashDispenser.Dispense* command specified that the amount has to be presented in multiple bunches, the completion message includes details about remaining bunches. The *additionalBunches* property specifies whether there are any additional bunches to be dispensed to the customer and the number of outstanding present operations. On the last present operation *additionalBunches* is set to "0" or omitted.

If the dispense operation is protected by end-to-end security then the device will track the total value of cash presented. Once the value of cash that has been dispensed and presented reaches the value of the token, the command nonce stored in the device will be cleared. This has the effect of making any existing tokens invalid so that they can't be used again. No more cash can be dispensed until a new command nonce is read and a new token is generated.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"position": "outDefault"	string	
}		

Properties

timeout

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

default: 0

position

Supplies the output position as one of the following values. Supported positions are reported in Common.Capabilities.

- `outDefault` - Default output 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.

default: "outDefault"

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "shutterNotOpen",	string	
"position": "inLeft",	string	
"additionalBunches": "1"	string	

Payload (version 1.0)	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. 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 Storage.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

Supplies the input or output position as one of the following values. If not specified, the default position applies. Supported positions are reported in Common.Capabilities.

- `inDefault` - Default input position.
- `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.
- `outDefault` - Default output 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.

additionalBunches

Specifies how many more bunches will be required to present the request. Following values are possible:

- `<number>` - The number of additional bunches to be presented.
- `unknown` - More than one additional bunch is required but the precise number is unknown.

Property value constraints:

```
pattern: ^unknown$|^[0-9]*$
```

default: "0"

Event Messages

- Storage.StorageThresholdEvent
- CashManagement.ItemsTakenEvent
- CashManagement.InfoAvailableEvent

- CashManagement.ItemsPresentedEvent

7.2.7 CashDispenser.Reject

This command will move items from the intermediate stacker to a *reject* storage unit. The storage unit's counts are incremented by the number of items that were or 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 storage unit counts may be unreliable.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

timeout

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

default: 0

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "cashUnitError"	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. Following values are possible:

- `cashUnitError` - A storage unit caused a problem. A Storage.StorageErrorEvent will be posted with the details.
- `noItems` - There were no items to reject.
- `exchangeActive` - The device is in an exchange state (see Storage.StartExchange).

Event Messages

- Storage.StorageThresholdEvent
- Storage.StorageErrorEvent
- CashManagement.InfoAvailableEvent

7.2.8 CashDispenser.SetMixTable

This command is used to set up the mix table specified by the *mixNumber*. Mix tables are persistent and are available to all applications in CashDispenser.Dispense and CashDispenser.Denominate commands. If mix table specified by the *mixNumber* already exists then the information is overwritten with the new information.

A mix specifies how a given requested amount is composed of a set of cash items, for example USD 100 could be 5 x USD 20 or 10 x USD 10. A mix table specifies multiple mixes. An amount can be specified multiple times to include different combinations of cash items, if an amount is specified more than once the Service will attempt to denominate or dispense the first amount in the table. If this mix is not possible (e.g., because of a storage unit failure) the Service will search for the first mix which is possible. The Service can only dispense amounts which are explicitly mentioned in the mix table.

Available mixes are reported by CashDispenser.GetMixTypes and the details of a stored mix table can be queried using CashDispenser.GetMixTable.

Command Message

Payload (version 1.0)	Type	Required

{
  "timeout": 5000,
  "mixNumber": 21,
  "name": "House mix 21",
  "mixRows": [{
  "amount": 0.30,
  "mix": [{
  "value": 0.05,
  "count": 6
  }]
  }]
}		

Properties

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

mixNumber
Number identifying the house mix table.
Property value constraints:
`minimum: 1`

name
Name of the house mix table.

mixRows
Array of rows of the mix table.

mixRows/amount
Absolute value of the amount denominated by this mix row.
Property value constraints:
`minimum: 0`

Properties
mixRows/mix The items used to create *amount*. Each element in this array defines the quantity of a given item used to create the mix. An example showing how 0.30 can be broken down would be: `[ { "value": 0.05, "count": 2 }, { "value": 0.10, "count": 2 } ]`
mixRows/mix/ The quantity of a given absolute value of cash items contained in the mix.
mixRows/mix/value The absolute value of a single cash item. Property value constraints: `minimum: 0`
mixRows/mix/count The number of items of *value* contained in the mix. Property value constraints: `minimum: 1`

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	string	
`"errorCode": "invalidMixNumber"`	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. Following values are possible: • `invalidMixNumber` - The *mixNumber* is invalid. • `invalidMixTable` - The contents of at least one of the defined rows of the mix table is incorrect.

Event Messages

None

7.2.9 CashDispenser.TestCashUnits

This command is used to test cash dispense storage units following replenishment. The command payload specifies where items dispensed as a result of this command should be moved to. The operation performed to test the storage units is vendor dependent.

All storage units which match the following criteria are tested.

- cashOut is true
- status is *ok*
- replenishmentStatus is not *empty*
- appLockOut is false

If the hardware is able to do so tests are continued even if an error occurs while testing one of the storage 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 storage units could be tested and a dispense was possible from at least one of the storage units.

A Storage.StorageErrorEvent will be sent for any *cashOut* unit which cannot be tested or which failed the test. If no storage units could be tested or no storage units are testable then a *cashUnitError* code will be returned and *Storage.StorageErrorEvent* events generated for every storage unit that encountered a problem.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"unit": "unit1",	string	
"retractArea": {	object	
"outputPosition": "outDefault",	string	
"retractArea": "retract",	string	
"index": 0	integer	
},		
"outputPosition": "outDefault"	string	
}		

Properties

timeout

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

default: 0

unit

Specifies the object name (as stated by the Storage.GetStorage command) of the single unit to be used for the storage of any items found.

Property value constraints:

```
pattern: ^unit[0-9A-Za-z]+$
```

retractArea

This property is used if items are to be moved to internal areas of the device, including storage units, the intermediate stacker, or the transport.

Properties
retractArea/outputPosition Supplies the output position as one of the following values. Supported positions are reported in Common.Capabilities. • `outDefault` - Default output 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. default: "outDefault"
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 storage unit. • `transport` - Retract the items to the transport. • `stacker` - Retract the items to the intermediate stacker area. • `reject` - Retract the items to a reject storage unit. • `itemCassette` - Retract the items to the storage units which would be used during a Cash In transaction including recycling storage units. • `cashIn` - Retract the items to the storage units which would be used during a Cash In transaction but not including recycling storage units.
retractArea/index If *retractArea* is set to *retract* this property defines the position inside the retract storage units into which the cash is to be retracted. *index* starts with a value of 1 for the first retract position and increments by one for each subsequent position. If there are several retract storage units (of type *retractCassette* in Storage.GetStorage), *index* would be incremented from the first position of the first retract storage unit to the last position of the last retract storage unit. The maximum value of *index* is the sum of *maximum* of each retract storage unit. If *retractArea* is not set to *retract* the value of this property is ignored.

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	string	
`"errorCode": "cashUnitError"`	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.

Properties
errorCode Specifies the error code if applicable. Following values are possible: • `cashUnitError` - A storage unit caused a problem that meant all storage units could not be tested or no storage units were testable. One or more Storage.StorageErrorEventevents 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 storage unit number specified is not valid. • `exchangeActive` - The device is in an exchange state (see Storage.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.

Event Messages

- Storage.StorageThresholdEvent
- Storage.StorageErrorEvent
- CashManagement.ItemsTakenEvent
- Storage.StorageChangedEvent
- CashManagement.NoteErrorEvent
- CashManagement.InfoAvailableEvent

7.2.10 CashDispenser.Count

This command empties the specified storage unit(s). All items dispensed from the 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 command execution the storage unit(s) counts are reset.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"unit": "unit1",	string	
"position": "outDefault"	string	
}		

Properties

timeout

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

default: 0

unit

Specifies the unit to empty or that all units are to be emptied. Following values are possible:

- `all` - All units are to be emptied.
- `<storage unit identifier>` - The storage unit to be emptied as identifier.

Property value constraints:

```
pattern: ^all$|^unit[0-9A-Za-z]+$
```

position

Supplies the output position as one of the following values. Supported positions are reported in Common.Capabilities.

- `outDefault` - Default output 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.

default: "outDefault"

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "cashUnitError",	string	
"countedCashUnits": {	object	
"*unit1*": {	object	

Payload (version 1.0)	Type	Required
"dispensed": 100,	integer	
"counted": 100,	integer	
"replenishmentStatus": "ok",	string	
"status": "ok"	string	
},		
"*unit2": {*	object	
See unit1 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. Following values are possible:

- `cashUnitError` - A storage unit caused a problem. A Storage.StorageErrorEvent 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 Storage.StartExchange).

countedCashUnits

List of counted storage unit objects.

countedCashUnits/unit1 (example name)

Counted storage unit object. Object name is the same as used in Storage.GetStorage.

Property name constraints:

```
pattern: ^unit[0-9A-Za-z]+$
```

countedCashUnits/unit1/dispensed

The number of items that were dispensed during the emptying of the storage unit.

Property value constraints:

```
minimum: 1
```

countedCashUnits/unit1/counted

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

Property value constraints:

```
minimum: 1
```

Properties
countedCashUnits/unit1/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. This is based on hardware detection, either on sensors or counts. • `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. If the storage unit has hardwareSensors, this state is not set by counts.
countedCashUnits/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.

Event Messages

- Storage.StorageErrorEvent
- CashManagement.ItemsTakenEvent
- CashManagement.ItemsPresentedEvent
- CashManagement.NoteErrorEvent
- CashManagement.InfoAvailableEvent

7.2.11 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 this command can be used to improve transaction performance.

If this command is supported 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 mechanical 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 (version 1.0)	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"action": "start"`	string	
`}`		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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 (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available"`	string	
`}`		

Properties

completionCode

The completion code.

errorDescription

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

Event Messages

None

7.3 Event Messages

7.3.1 CashDispenser.DelayedDispenseEvent

This event is generated if the start of a dispense operation has been delayed.

Event Message

Payload (version 1.0)	Type	Required
{		
"delay": 0.1	number	
}		
Properties		
delay The time in seconds by which the dispense operation will be delayed.		

7.3.2 CashDispenser.StartDispenseEvent

This event is generated when a delayed dispense operation begins.

Event Message

Payload (version 1.0)	Type	Required
{		
}		

7.3.3 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 payload 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.

Event Message

Payload (version 1.0)	Type	Required
{		
"currencies": {	object	
"*EUR*": 10,	number	
"*USD*": 10	number	
},		
"values": {	object	
"*unit1*": 5,	integer	
"*unit2*": 5	integer	
},		
"cashBox": {	object	
See currencies properties.		
}		
}		

Properties

Specifies a denomination or a denomination request.

currencies

List of currency and amount combinations for denomination requests or output. There will be one entry for each currency in the denomination. This list can be omitted on a request if *values* specifies the entire request.

currencies/EUR (example name)

The absolute amount to be or which has been denominated or dispensed of the currency. The property name is the ISO 4217 currency identifier [Ref. cashdispenser-1]. The property value can include a decimal point to specify fractions of the currency, for example coins.

Property name constraints:

```
pattern: ^[A-Z]{3}$
```

Property value constraints:

```
minimum: 0.001
```

values

This list specifies the number of items to take or which have been taken from the storage units. If specified in a request, the output denomination must include these items.

The property name is storage unit object name as stated by the Storage.GetStorage command. The value of the entry is the number of items to take from that unit.

values/unit1 (example name)

The number of items have been dispensed from the specified storage unit to meet the request.

Property name constraints:

```
pattern: ^unit[0-9A-Za-z]+$
```

Property value constraints:

```
minimum: 1
```

Properties
cashBox Only applies to Teller Dispensers. Amount to be paid from the teller’s cash box.

8. Cash Acceptor Interface

This chapter defines the Cash Acceptor interface functionality and messages.

This specification describes the functionality of an XFS4IoT compliant Cash Acceptor interface. It defines the interface-specific commands that can be issued to the service using the WebSocket endpoint.

Persistent values are maintained through power failures, open sessions, close sessions 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.

8.1 Command Messages

8.1.1 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 CashAcceptor.CashInStart command.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"status": "ok",	string	
"numOfRefused": 0,	integer	
"noteNumberList": {	object	
"unrecognized": 0,	integer	
"*type20USD1*": {	object	
"fit": 0,	integer	
"unfit": 0,	integer	
"suspect": 0,	integer	
"counterfeit": 0,	integer	
"inked": 0	integer	
},		
"*type50USD1*": {	object	
See type20USD1 properties.		
}		
}		
}		

Properties

completionCode
The completion code.

Properties
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
status Status of the currently active or most recently ended cash-in transaction. The following values are possible: • `ok` - The cash-in transaction is complete and has ended with CashAcceptor.CashInEnd. • `rollback` - The cash-in transaction ended with CashAcceptor.CashInRollback. • `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 CashManagement.Retract. • `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 CashManagement.Reset.
numOfRefused Specifies the number of items refused during the currently active or most recently ended cash-in transaction period. Property value constraints: `minimum: 0`
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. Includes any identified notes.
noteNumberList/unrecognized Count of unrecognized items handled by the cash interface.
noteNumberList/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification.
noteNumberList/type20USD1/fit Count of genuine cash items which are fit for recycling.
noteNumberList/type20USD1/unfit Count of genuine cash items which are unfit for recycling.
noteNumberList/type20USD1/suspect Count of suspected counterfeit cash items.
noteNumberList/type20USD1/counterfeit Count of counterfeit cash items.
noteNumberList/type20USD1/inked Count of cash items which have been identified as ink stained.

Event Messages

None

8.1.2 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 (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		
Properties		
timeout Timeout in milliseconds for the command to complete. If set to 0, the command will not timeout but can be canceled. default: 0		

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"posCapabilities": [{	array (object)	
"position": "inLeft",	string	
"usage": {	object	
"in": false,	boolean	
"refuse": false,	boolean	
"rollback": false	boolean	
},		
"shutterControl": false,	boolean	
"itemsTakenSensor": false,	boolean	
"itemsInsertedSensor": false,	boolean	
"retractAreas": {	object	
"retract": false,	boolean	
"reject": false,	boolean	
"transport": false,	boolean	
"stacker": false,	boolean	
"billCassettes": false,	boolean	
"cashIn": false	boolean	
},		
"presentControl": false,	boolean	
"preparePresent": false	boolean	
}]		
}		

Properties
completionCode The completion code.
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.
posCapabilities Array of position capabilities for all positions configured in this service.
posCapabilities/position Supplies the input or output position as one of the following values. If not specified, the default position applies. Supported positions are reported in Common.Capabilities. • `inDefault` - Default input position. • `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. • `outDefault` - Default output 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 a position is used to input, 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 true the shutter is controlled implicitly by the Service. If false the shutter must be controlled explicitly by the application using the CashManagement.OpenShutter and CashManagement.CloseShutter commands. In either case the CashAcceptor.PresentMedia command may be used if *presentControl* is false. The *shutterControl* field is always 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 true the service generates an accompanying CashManagement.ItemsTakenEvent. If 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 true the service generates an accompanying CashManagement.ItemsInsertedEvent. If false this event is not generated. This field relates to all input positions.

Properties
posCapabilities/retractAreas Specifies the areas to which items may be retracted from this position. This is not reported if the device cannot retract.
posCapabilities/retractAreas/retract Items may be retracted to a retract storage unit.
posCapabilities/retractAreas/reject Items may be retracted to a reject storage 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 storage units.
posCapabilities/retractAreas/cashIn Items may be retracted to a cash-in storage unit.
posCapabilities/presentControl Specifies how the presenting of media items is controlled. If 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 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 false then items to be removed are moved to the output position as part of the relevant command. e.g. *CashManagement.OpenShutter*, *CashAcceptor.PresentMedia* or *CashAcceptor.CashInRollback*. If true then items are moved to the output position using the CashAcceptor.PreparePresent command.

Event Messages

None

8.1.3 CashAcceptor.GetReplenishTarget

This command is used to determine which storage units can be specified as targets for a given source storage 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 unit.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"source": "unit2"	string	
}		

Properties

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

source
The name of the storage unit (as stated by the Storage.GetStoragecommand) which would be used as the source of the replenishment operation.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"targets": [{	array (object)	
"target": "unit1"	string	
}]		
}		

Properties

completionCode
The completion code.

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

targets
Array of all suitable replenish targets. Empty if no suitable target was found.

targets/target
The name of the storage unit (as stated by the Storage.GetStoragecommand) that can be used as a target.
Property value constraints:

```
pattern: ^unit[0-9A-Za-z]+$
```

Event Messages

None

8.1.4 CashAcceptor.GetDeviceLockStatus

This command is used to retrieve the lock/unlock statuses of the CashAcceptor device and each of its storage units. This is only supported if the physical locking and unlocking of the device or the storage units is supported.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"deviceLockStatus": "lock",	string	
"unitLock": [{	array (object)	
"storageUnit": "unit1",	string	
"unitLockStatus": "lock"	string	
}]		
}		

Properties

completionCode
The completion code.

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

deviceLockStatus
Specifies the physical lock/unlock status of the CashAcceptor device. The 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 reporting the physical lock/unlock status of the device.

unitLock
Array specifying the physical lock/unlock status of storage units. Units that do not support the physical lock/unlock control are not contained in the array. If there are no units that support physical lock/unlock control this will be empty.

Properties
unitLock/storageUnit Object name of the storage unit as stated by Storage.GetStorage. Property value constraints: `pattern: ^unit[0-9A-Za-z]+$`
unitLock/unitLockStatus Specifies the physical lock/unlock status of storage units supported. The following values are possible: • `lock` - The storage unit is physically locked. • `unlock` - The storage unit is physically unlocked. • `lockUnknown` - Due to a hardware error or other condition, the physical lock/unlock status of the storage unit cannot be determined.

Event Messages

None

8.1.5 CashAcceptor.GetDepleteSource

This command is used to determine which storage units can be specified as source storage units for a given target storage 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 storage unit.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"cashUnitTarget": "unit2"	string	
}		

Properties

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

cashUnitTarget
Object name of the storage unit (as stated by the Storage.GetStoragecommand) which would be used as the target of the depletion operation.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"depleteSources": [{	array (object)	
"cashUnitSource": "unit1"	string	
}]		
}		

Properties

completionCode
The completion code.

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

depleteSources
Array of all suitable deplete sources. Empty if no suitable source was found.

depleteSources/cashUnitSource
The name of the storage unit (as stated by the Storage.GetStoragecommand) that can be used as a source.
Property value constraints:

```
pattern: ^unit[0-9A-Za-z]+$
```

Event Messages

None

8.1.6 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: CashAcceptor.CashIn, CashAcceptor.CashInRollback, CashAcceptor.PreparePresent, CashAcceptor.PresentMedia, CashManagement.OpenShutter (In the case of returning multiple bunches)

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"position": "outDefault",	string	
"presentState": "presented",	string	
"additionalBunches": "none",	string	
"bunchesRemaining": 0,	integer	
"returnedItems": {	object	
"unrecognized": 0,	integer	
"*type20USD1*": {	object	
"fit": 0,	integer	
"unfit": 0,	integer	
"suspect": 0,	integer	
"counterfeit": 0,	integer	
"inked": 0	integer	
},		
"*type50USD1*": {	object	
See type20USD1 properties.		
}		
},		
"totalReturnedItems": {	object	
See returnedItems properties.		
},		

Payload (version 1.0)	Type	Required
"remainingItems": {	object	
See returnedItems properties.		
}		
}		

Properties

completionCode

The completion code.

errorDescription

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

position

Supplies the output position as one of the following values. Supported positions are reported in Common.Capabilities.

- `outDefault` - Default output 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.

default: "outDefault"

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. The 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. The 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. This property is omitted if any of the following are true:

- If the number of additional bunches is at least one, but the precise number is unknown.
- *additionalBunches* is not `oneMore`.

returnedItems

Array holding a list of counts of banknotes which have been moved to the output position as a result of the most recent operation.

returnedItems/unrecognized

Count of unrecognized items handled by the cash interface.

returnedItems/type20USD1 (example name)

Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification.

Properties
returnedItems/type20USD1/fit Count of genuine cash items which are fit for recycling.
returnedItems/type20USD1/unfit Count of genuine cash items which are unfit for recycling.
returnedItems/type20USD1/suspect Count of suspected counterfeit cash items.
returnedItems/type20USD1/counterfeit Count of counterfeit cash items.
returnedItems/type20USD1/inked Count of cash items which have been identified as ink stained.
totalReturnedItems Array of cumulative counts of banknotes which have been moved to the output position. This value will be reset when a *CashAcceptor.CashInStart*, *CashAcceptor.CashIn*, *CashAcceptor.CashInEnd*, *CashManagement.Retract*, *CashManagement.Reset* or *CashAcceptor.CashInRollback*command is executed.
remainingItems Array of counts of banknotes on the intermediate stacker or transport which have not been yet moved to the output position.

Event Messages

None

8.1.7 CashAcceptor.CashInStart

Before initiating a cash-in operation, an application must issue this 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, CashManagement.Retract or CashManagement.Reset command is sent. Where shutterControl is false this command precedes any explicit operation of the shutters.

If an application wishes to determine where the notes went during a transaction it can execute a Storage.GetStorage 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.

If supported by cashInLimit, an individual cash-in transaction can be limited to a maximum number of items (*totalItemsLimit*) or a maximum amount (*amountLimit*). If not supported or specified, the number of items accepted in the transaction is limited by the capacity of the intermediateStacker. Any limitations specified by these parameters only apply to the individual cash-in transaction; subsequent transactions are not affected. The following table shows some examples of how the transaction can be limited.

Transaction limits	***totalItemsLimit***	***amountLimit***
EUR 100 or GBP 200 or USD 500 Maximum number of items allowed limited by physical capability.	0	EUR 100 GBP 200 USD 500
EUR 100 or GBP 200, USD refused Maximum 50 items allowed.	50	EUR 100 GBP 200
USD 500, no limit on GBP, other currencies refused Maximum number of items allowed limited by physical capability.	0	GBP 0 USD 500
EUR limited by physical capability of the device. Other currencies refused.	0	EUR 0
EUR limited by physical capability of the device. GBP 100, USD refused.	0	EUR 0 GBP 100

Command Message

Payload (version 1.0)	**Type**	**Required**
{		
"timeout": 5000,	integer	
"tellerID": 0,	integer	
"useRecycleUnits": true,	boolean	
"outputPosition": "outDefault",	string	
"inputPosition": "inDefault",	string	
"totalItemsLimit": 0,	integer	
"amountLimit": [{	array (object)	
"currency": "USD",	string	
"value": 20	number	
}]		
}		

Properties

timeout

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

default: 0

Properties
tellerID Identification of teller. This field is not applicable to Self-Service devices and should be omitted. Property value constraints: `minimum: 0` default: 0
useRecycleUnits Specifies whether or not the recycle storage 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 storage units or the hardware does not support this. default: true
outputPosition Supplies the output position as one of the following values. Supported positions are reported in Common.Capabilities. • `outDefault` - Default output 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. default: "outDefault"
inputPosition Supplies the input position as one of the following values. Supported positions are reported in Common.Capabilities. • `inDefault` - Default input position. • `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. default: "inDefault"
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 0, there will be no limit on the number of items. This limitation can only be used if byTotalItems is true. Property value constraints: `minimum: 0` default: 0
amountLimit If specified, provides a list of the maximum amount of one or more currencies to be accepted during the cash-in transaction. This limitation can only be used if byAmount is true. If not specified, no currency specific limit is placed on the transaction. If specified for one currency and the device can handle multiple currencies in a single cash-in transaction, any currencies not defined in this array are refused.
amountLimit/currency ISO 4217 currency identifier [Ref. cashmanagement-1]. Property value constraints: `pattern: ^[A-Z]{3}$`

Properties
amountLimit/value The maximum absolute value of the specified currency which can be accepted in the cash-in transaction. If 0, there is no amount limit applied to the currency. Property value constraints: `minimum: 0` default: 0

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	string	
`"errorCode": "invalidTellerId"`	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: • `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 this command.

Event Messages

None

8.1.8 CashAcceptor.CashIn

This command moves items into the cash device from an input position.

On devices with implicit shutter control, the CashAcceptor.InsertItemsEvent 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.InputRefuseEvent events will be sent to report the rejection. See also the paragraph below about 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 genuine items (see Note Classification) to be moved there after validation. Counterfeit, suspect or inked items may also be moved to the cash-in stacker, but some devices may immediately move them to a designated storage unit. Items on the stacker will remain there until the current cash-in transaction is either cancelled by the CashAcceptor.CashInRollback command or confirmed by the CashAcceptor.CashInEnd command. These commands will cause any non-genuine items on the cash-in stacker to be moved to the appropriate storage unit. If there is no cash-in stacker then this command will move items directly to the storage units and the *CashAcceptor.CashInRollback* command will not be supported. Storage unit information will be updated accordingly whenever notes are moved to a storage unit during this command.

Note that the acceptor status field may change value during a cash-in transaction. If media has been retained to storage 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 errorCode `cashUnitError`.

The shutterControl field will determine whether the shutter is controlled implicitly by this command or whether the application must explicitly open and close the shutter using the CashManagement.OpenShutter, CashManagement.CloseShutter or CashAcceptor.PresentMedia commands. 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 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 *CashManagement.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.SubCashInEvent 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 CashManagement.ItemsPresentedEvent will be generated.

If *shutterControl* is false, and a single bunch of items is returned then this command will complete without generating a *CashManagement.ItemsPresentedEvent*, instead the event will be generated by the subsequent *CashManagement.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 *CashManagement.ItemsPresentedEvent* has been generated. For these devices *shutterControl* and *presentControl* fields of the *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 Common.Cancel before the final bunch is presented so that after the completion of this command the CashManagement.Retract or CashManagement.Reset command can be used to move the remaining bunches, although the ability to do this will be hardware dependent.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "cashUnitError",	string	
"unrecognized": 0,	integer	
"*type20USD1*": {	object	
"fit": 0,	integer	
"unfit": 0,	integer	
"suspect": 0,	integer	
"counterfeit": 0,	integer	
"inked": 0	integer	
},		
"*type50USD1*": {	object	
See type20USD1 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.

Properties
errorCode Specifies the error code if applicable. The following values are possible: • `cashUnitError` - A problem occurred with a storage unit. A Storage.StorageErrorEvent 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.CashInStart 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 this command. • `foreignItemsDetected` - Foreign items have been detected inside the input position. • `shutterNotOpen` - Shutter failed to open.
unrecognized Count of unrecognized items handled by the cash interface.
type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification.
type20USD1/fit Count of genuine cash items which are fit for recycling.
type20USD1/unfit Count of genuine cash items which are unfit for recycling.
type20USD1/suspect Count of suspected counterfeit cash items.
type20USD1/counterfeit Count of counterfeit cash items.
type20USD1/inked Count of cash items which have been identified as ink stained.

Event Messages

- Storage.StorageErrorEvent
- Storage.StorageThresholdEvent
- CashAcceptor.InputRefuseEvent
- CashManagement.NoteErrorEvent
- CashAcceptor.SubCashInEvent
- CashManagement.ItemsInsertedEvent
- CashManagement.ItemsTakenEvent
- CashManagement.ItemsPresentedEvent
- CashManagement.InfoAvailableEvent
- CashAcceptor.InsertItemsEvent

8.1.9 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 storage units.

The cash-in transaction is ended even if this command does not complete successfully.

In the special case where all the items inserted by the customer are classified as counterfeit and/or suspect items and the Service is configured to automatically retain these item types then the command will complete with *success* even if the hardware may have already moved the counterfeit and/or suspect items to their respective storage units on the *CashAcceptor.CashIn* command and there are no items on the stacker at the start of the command. This allows the location of the notes retained to be reported in the output parameter. If no items are available for cash-in for any other reason, the *noItems* error code is returned.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		
Properties		
timeout Timeout in milliseconds for the command to complete. If set to 0, the command will not timeout but can be canceled. default: 0		

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "cashUnitError",	string	
"storage": {	object	
"*unit1*": {	object	
"retractOperations": 0,	integer	
"deposited": {	object	
"unrecognized": 0,	integer	
"*type20USD1*": {	object	
"fit": 0,	integer	
"unfit": 0,	integer	
"suspect": 0,	integer	
"counterfeit": 0,	integer	
"inked": 0	integer	
},		
"*type50USD1*": {	object	
See type20USD1 properties.		
}		
},		

Payload (version 1.0)	Type	Required
"retracted": {	object	
See deposited properties.		
},		
"rejected": {	object	
See deposited properties.		
},		
"distributed": {	object	
See deposited properties.		
},		
"transport": {	object	
See deposited properties.		
}		
},		
"*unit2*": {	object	
See unit1 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:

- `cashUnitError` - A problem occurred with a storage unit. A Storage.StorageErrorEvent 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 this command.

storage

List of storage units that have taken items, and the type of items they have taken, during the current transaction. This only contains data related to the current transaction.

storage/unit1 (example name)

List of items moved to this storage unit by this transaction or command. The property name is the same as reported by Storage.GetStorage.

Property name constraints:

```
pattern: ^unit[0-9A-Za-z]+$
```

storage/unit1/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.

Properties
storage/unit1/deposited The items deposited in the storage unit during a Cash In transaction.
storage/unit1/deposited/unrecognized Count of unrecognized items handled by the cash interface.
storage/unit1/deposited/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification.
storage/unit1/deposited/type20USD1/fit Count of genuine cash items which are fit for recycling.
storage/unit1/deposited/type20USD1/unfit Count of genuine cash items which are unfit for recycling.
storage/unit1/deposited/type20USD1/suspect Count of suspected counterfeit cash items.
storage/unit1/deposited/type20USD1/counterfeit Count of counterfeit cash items.
storage/unit1/deposited/type20USD1/inked Count of cash items which have been identified as ink stained.
storage/unit1/retracted The items retracted into 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/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/distributed The items deposited in this storage unit originating from another storage unit but not rejected.
storage/unit1/transport The items which were intended to be deposited in this storage unit but are not yet deposited. Typical use case for this property is tracking items after a jam during CashAcceptor.CashInEnd. This is not reset if initial is set for this unit by Storage.GetStorage.

Event Messages

- Storage.StorageChangedEvent
- Storage.StorageErrorEvent
- Storage.StorageThresholdEvent
- CashManagement.InfoAvailableEvent
- CashManagement.NoteErrorEvent
- CashManagement.ItemsTakenEvent
- CashManagement.ItemsPresentedEvent

8.1.10 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 will determine whether the shutter is controlled implicitly by this command or whether the application must explicitly open and close the shutter using the CashManagement.OpenShutter, CashManagement.CloseShutter or CashAcceptor.PresentMedia commands. 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 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 *CashManagement.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.

In the special case where all the items inserted by the customer are classified as counterfeit and/or suspect, and the Service is configured to automatically retain these item types, then the command will complete with *success* even though no items are returned to the customer. This allows the location of the notes retained to be reported in the output parameter. The application can tell if items have been returned or not via the CashManagement.ItemsPresentedEvent. This event will be generated before the command completes when items are returned. This event will not be generated if no items are returned. If no items are available to rollback for any other reason, the *noItems* error code is returned.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "cashUnitError",	string	
"storage": {	object	
"*unit1*": {	object	
"retractOperations": 0,	integer	
"deposited": {	object	
"unrecognized": 0,	integer	

Payload (version 1.0)	Type	Required
"*type20USD1*": {	object	
"fit": 0,	integer	
"unfit": 0,	integer	
"suspect": 0,	integer	
"counterfeit": 0,	integer	
"inked": 0	integer	
},		
"*type50USD1": {*	object	
See type20USD1 properties.		
}		
},		
"retracted": {	object	
See deposited properties.		
},		
"rejected": {	object	
See deposited properties.		
},		
"distributed": {	object	
See deposited properties.		
},		
"transport": {	object	
See deposited properties.		
}		
},		
"*unit2": {*	object	
See unit1 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.

Properties
errorCode Specifies the error code if applicable. The following values are possible: • `cashUnitError` - A problem occurred with a storage unit. A Storage.StorageErrorEvent will be sent with the details. • `shutterNotOpen` - The 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.
storage List of storage units that have taken items, and the type of items they have taken, during the current transaction. This only contains data related to the current transaction.
storage/unit1 (example name) List of items moved to this storage unit by this transaction or command. The property name is the same as reported by Storage.GetStorage. Property name constraints: `pattern: ^unit[0-9A-Za-z]+$`
storage/unit1/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/deposited The items deposited in the storage unit during a Cash In transaction.
storage/unit1/deposited/unrecognized Count of unrecognized items handled by the cash interface.
storage/unit1/deposited/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification.
storage/unit1/deposited/type20USD1/fit Count of genuine cash items which are fit for recycling.
storage/unit1/deposited/type20USD1/unfit Count of genuine cash items which are unfit for recycling.
storage/unit1/deposited/type20USD1/suspect Count of suspected counterfeit cash items.
storage/unit1/deposited/type20USD1/counterfeit Count of counterfeit cash items.
storage/unit1/deposited/type20USD1/inked Count of cash items which have been identified as ink stained.
storage/unit1/retracted The items retracted into 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/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/distributed The items deposited in this storage unit originating from another storage unit but not rejected.

Properties
storage/unit1/transport The items which were intended to be deposited in this storage unit but are not yet deposited. Typical use case for this property is tracking items after a jam during CashAcceptor.CashInEnd. This is not reset if initial is set for this unit by Storage.GetStorage.

Event Messages

- Storage.StorageErrorEvent
- Storage.StorageThresholdEvent
- CashManagement.ItemsTakenEvent
- CashManagement.ItemsPresentedEvent
- CashManagement.InfoAvailableEvent

8.1.11 CashAcceptor.ConfigureNoteTypes

This command is used to change the note types the banknote reader should accept during cash-in. Only note types which are to be changed need to be specified in the command payload. If an unknown note type is given the completion code *unsupportedData* will be returned.

The values set by this command are persistent.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"items": [{	array (object)	
"item": "type20USD1",	string	
"enabled": true	boolean	
}]		
}		

Properties

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

items
An array which specifies which note types are to be disabled or re-enabled.

items/item
A cash item as reported by CashManagement.GetBankNoteTypes. This is not specified if the item was not identified as a cash item.
Property value constraints:

```
pattern: ^type[0-9A-Z]+$
```

items/enabled
If true the banknote reader will accept this note type during a cash-in operations. If false the banknote reader will refuse this note type, unless it must be retained by note classification rules.
default: true

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "exchangeActive"	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.

Properties
errorCode Specifies the error code if applicable. The 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

8.1.12 CashAcceptor.CreateSignature

This command is used to create a reference signature which 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 will determine whether the shutter is controlled implicitly by this command or whether the application must explicitly open and close the shutter using the CashManagement.OpenShutter, CashManagement.CloseShutter or CashAcceptor.PresentMedia commands. If *shutterControl* is false then this command does not operate the shutter in any way, and 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 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 *CashManagement.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 CashAcceptor.InsertItemsEvent 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 CashAcceptor.InputRefuseEvent will be sent and the command will complete. In this case, no note specific output properties will be returned.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

timeout

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

default: 0

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "tooManyItems",	string	
"noteType": "type20USD1",	string	
"orientation": "frontTop",	string	
"signature": "MAA5ADgANwA2ADUANAAz ..."	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: • `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.
noteType A cash item as reported by CashManagement.GetBankNoteTypes. This is not specified if the item was not identified as a cash item. Property value constraints: `pattern: ^type[0-9A-Z]+$`
orientation Specifies the note orientation. The following values are possible: • `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. • `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. • `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. • `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. • `unknown` - The orientation for the inserted note cannot be determined. • `notSupported` - The hardware is not capable to determine the orientation.

signature
Base64 encoded vendor specific signature data. If no signature is available or has not been requested then this is omitted.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

Event Messages

- CashAcceptor.InputRefuseEvent
- CashManagement.ItemsInsertedEvent
- CashManagement.ItemsTakenEvent
- CashManagement.ItemsPresentedEvent
- CashManagement.NoteErrorEvent
- CashAcceptor.InsertItemsEvent
- CashManagement.InfoAvailableEvent

8.1.13 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"loadAlways": false	boolean	
}		

Properties

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

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 (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "exchangeActive",	string	
"rebootNecessary": 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:

- `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 storage 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

8.1.14 CashAcceptor.CompareSignature

This command is used to compare the signatures of a reference item with the available signatures of the cash-in transactions.

The reference signatures are created by the CashAcceptor.CreateSignature command.

The transaction signatures are obtained through the CashManagement.GetItemInfo command.

The signatures (1 to 4) of the reference banknote are typically the signatures of the 4 orientations of the banknote.

The *CashAcceptor.CompareSignature* 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 does not support the confidence level factor, it returns a single indication to the best matching signature with the confidence level factor set to 0.

If the comparison completed with no matching signatures found then the command returns "ok" with *signaturesIndex* empty.

This command must be used outside of cash-in transactions and outside of the exchange state.

Due to the potential for signatures to be large, as well as the possibility that it may be necessary to compare the reference signature with a large number of signatures, applications should be aware of the amount of data passed as input to this command. In some cases, it may be necessary to execute this command more than once, with subsets of the total signatures, and then afterward compare the results from each execution.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"referenceSignatures": [{	array (object)	
"noteType": "type20USD1",	string	
"orientation": "frontTop",	string	
"signature": "MAA5ADgANwA2ADUANAAz ..."	string	
}],		
"signatures": [{	array (object)	
See referenceSignatures properties.		
}]		
}		

Properties

timeout

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

default: 0

referenceSignatures

Array of Signature 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.

referenceSignatures/noteType

A cash item as reported by CashManagement.GetBankNoteTypes. This is not specified if the item was not identified as a cash item.

Property value constraints:

```
pattern: ^type[0-9A-Z]+$
```

Properties
referenceSignatures/orientation Specifies the note orientation. The following values are possible: • `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. • `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. • `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. • `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. • `unknown` - The orientation for the inserted note cannot be determined. • `notSupported` - The hardware is not capable to determine the orientation.

referenceSignatures/signature
Base64 encoded vendor specific signature data. If no signature is available or has not been requested then this is omitted.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

signatures Array of Signature structures. Each structure represents a signature from the cash-in transactions, to be compared with the reference signatures in *referenceSignatures*. At least one signature must be provided. If there are no signatures provided (this array is missing or empty) the command returns an *invalidData* error.

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	string	
`"errorCode": "cashInActive",`	string	
`"signaturesIndex": [{`	array (object)	
`"index": 0,`	integer	
`"confidenceLevel": 95,`	integer	
`"comparisonData": "Example comparison data."`	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.

Properties
errorCode Specifies the error code if applicable. The following values are possible: • `cashInActive` - A cash-in transaction is active. This device requires that no cash-in transaction is active in order to perform the command. • `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.
signaturesIndex Array of compare results. This array is empty when the compare operation completes with no matches found. If there are matches found, *signaturesIndex* contains the indices of the matching signatures from the input parameter *signatures*. If there is a match found but the Service does not support the confidence level factor, *signaturesIndex*contains a single index with confidenceLevel set to 0.
signaturesIndex/index Specifies the index (0 to #*signatures* - 1) of the matching signature from the input parameter *signatures*.

signaturesIndex/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 0 if the Service does not support the confidence level factor.
Property value constraints:
`minimum: 0`
`maximum: 100`

signaturesIndex/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

8.1.15 CashAcceptor.Replenish

This command replenishes items from a single storage unit to multiple storage 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 storage unit to a target storage unit. This is especially applicable if a replenishment storage unit is used for the replenishment and can help to minimize manual replenishment operations.

The CashAcceptor.GetReplenishTarget command can be used to determine what storage units can be specified as target storage units for a given source storage unit. Any items which are removed from the source cash unit that are not of the correct currency and value for the target storage unit during execution of this command will be returned to the source storage unit.

The counts returned with the Storage.GetStorage 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"source": "unit1",	string	
"replenishTargets": [{	array (object)	
"target": "unit1",	string	
"numberOfItemsToMove": 100	integer	
}]		
}		

Properties

timeout

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

default: 0

source

Name of the storage unit (as stated by the Storage.GetStoragecommand) from which items are to be removed.

Property value constraints:

```
pattern: ^unit[0-9A-Za-z]+$
```

replenishTargets

Array of target elements specifying how many items are to be moved and to where. There must be at least one array element.

replenishTargets/target

Object name of the storage unit (as stated by the Storage.GetStoragecommand) to which items are to be moved.

Property value constraints:

```
pattern: ^unit[0-9A-Za-z]+$
```

replenishTargets/numberOfItemsToMove

The number of items to be moved to the target storage unit. If 0, all items will be moved. Any items which are removed from the source storage unit that are not of the correct currency and value for the target storage unit during execution of this command will be returned to the source storage unit.

Property value constraints:

```
minimum: 0
```

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "cashUnitError",	string	
"numberOfItemsRemoved": 20,	integer	
"numberOfItemsRejected": 2,	integer	
"replenishTargetResults": [{	array (object)	
"target": "unit1",	string	
"cashItem": "type20USD1",	string	
"numberOfItemsReceived": 20	integer	
}]		
}		

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:

- `cashUnitError` - A problem occurred with a storage unit. A Storage.StorageErrorEvent will be sent with the details. If appropriate a CashAcceptor.IncompleteReplenishEvent will also be sent.
- `invalidCashUnit` - The source or target storage unit specified is invalid for this operation. The CashAcceptor.GetReplenishTarget 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 storage unit including rejected items during execution of this command. Not specified if no items were removed.

Property value constraints:

```
minimum: 1
```

numberOfItemsRejected

Total number of items rejected during execution of this command. Not specified if no items were rejected.

Property value constraints:

```
minimum: 1
```

replenishTargetResults

Breakdown of which notes were moved and where they moved to. In the case where one note type has several releases and these are moved, or where items are moved from a multi denomination storage unit to a multi denomination storage unit, each target can receive several note types.

For example:

- If one single target was specified with the *replenishTargets* input structure, and this target received two different note types, then this property will have two elements.
- If two targets were specified and the first target received two different note types and the second target received three different note types, then this property will have five elements.

Properties
replenishTargetResults/target Name of the storage unit (as stated by the Storage.GetStoragecommand) to which items have been moved. Property value constraints: `pattern: ^unit[0-9A-Za-z]+$`
replenishTargetResults/cashItem A cash item as reported by CashManagement.GetBankNoteTypes. This is not specified if the item was not identified as a cash item. Property value constraints: `pattern: ^type[0-9A-Z]+$`
replenishTargetResults/numberOfItemsReceived Total number of items received in this target storage unit of the *cashItem* note type. Property value constraints: `minimum: 1`

Event Messages

- Storage.StorageErrorEvent
- Storage.StorageThresholdEvent
- CashManagement.NoteErrorEvent
- CashManagement.InfoAvailableEvent
- CashAcceptor.IncompleteReplenishEvent

8.1.16 CashAcceptor.CashUnitCount

This command counts the items in the storage unit(s). If it is necessary to move items internally to count them, the items should be returned to the unit from which they originated before completion of the command. If items could not be moved back to the storage unit they originated from and did not get rejected, the command will complete with an appropriate error.

During the execution of this command one Storage.StorageChangedEvent will be generated for each storage unit that has been counted successfully, or if the counts have changed, even if the overall command fails.

If an application wishes to determine where the notes went during the command it can execute a Storage.GetStorage before and after the transaction and then derive the difference.

This command is designed to be used on devices where the counts cannot be guaranteed to be accurate and therefore may need to be automatically counted periodically. Upon successful completion, for those storage units that have been counted, the counts are accurately reported with the *Storage.GetStorage* command.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"units": ["unit1", "unit2"]	array (string)	
}		

Properties

timeout

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

default: 0

units

Array containing the identifiers of the individual storage units to be counted. If an invalid storage unit is contained in this list, the command will fail with a *cashUnitError errorCode*.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "invalidCashUnit"	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.

Properties
errorCode Specifies the error code if applicable. The following values are possible: • `invalidCashUnit` - At least one of the storage units specified is either invalid or does not support being counted. No storage 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 storage unit from which they originated. • `countPositionNotEmpty` - A required internal position is not empty so a storage unit count is not possible. • `cashUnitError` - A storage unit caused a problem. A Storage.StorageErrorEvent will be posted with the details.

Event Messages

- Storage.StorageChangedEvent
- Storage.StorageErrorEvent
- Storage.StorageThresholdEvent
- CashManagement.NoteErrorEvent
- CashManagement.InfoAvailableEvent

8.1.17 CashAcceptor.DeviceLockControl

This command can be used to lock or unlock a CashAcceptor device or one or more storage units. [CashAcceptor.GetDeviceLockStatus] can be used to obtain the current lock state of any items which support locking.

During normal device operation the device and storage units will be locked and removal will not be possible. If supported, the device or storage 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 storage units are not physically removed from their normal operating position.

If the lock action is specified and the device or storage units are already locked, or if the unlock action is specified and the device or storage units are already unlocked then the action will complete successfully.

Once a storage unit has been removed and reinserted it may then have a *manipulated* status. This status can only be cleared by issuing a Storage.StartExchange / Storage.EndExchange command sequence.

The device and all storage units will also be locked implicitly as part of the execution of the *Storage.EndExchange* or the CashManagement.Reset command.

The normal command sequence is as follows:

1. *CashAcceptor.DeviceLockControl* command is executed to unlock the device and some or all of the storage units.
2. Optionally a cash-in transaction or a dispense transaction on a cash recycler device may be performed.
3. The operator was not required to remove any of the storage units, all storage units are still in their original position.
4. *CashAcceptor.DeviceLockControl* command is executed to lock the device and the storage units.

The relation of lock/unlock control with the *Storage.StartExchange* and the *Storage.EndExchange* commands is as follows:

1. *CashAcceptor.DeviceLockControl* command is executed to unlock the device and some or all of the storage units.
2. Optionally a CashAcceptor.CashInStart / CashAcceptor.CashIn / CashAcceptor.CashInEnd cash-in transaction or a CashDispenser.Dispense / CashDispenser.Present transaction on a cash recycler device may be performed.
3. The operator removes and reinserts one or more of the previously unlocked storage units. The associated Storage.StorageChangedEvent will be posted and after the reinsertion the storage unit will show the status *manualInsertion*.
4. *Storage.StartExchange* command is executed.
5. *Storage.EndExchange* command is executed. During this command execution the Service implicitly locks the device and all previously unlocked storage units. The status of the previously removed unit will be reset.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"deviceAction": "lock",	string	
"cashUnitAction": "lockAll",	string	
"unitLockControl": [{	array (object)	
"storageUnit": "unit1",	string	
"unitAction": "lock"	string	
}]		
}		

Properties
timeout Timeout in milliseconds for the command to complete. If set to 0, the command will not timeout but can be canceled. default: 0
deviceAction Specifies locking or unlocking the device in its normal operating position. The 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 storage units. The following values are possible: • `lockAll` - Locks all storage units supported. • `unlockAll` - Unlocks all storage units supported. • `lockIndividual` - Locks/unlocks storage units individually as specified in the *unitLockControl* parameter. • `noLockAction` - No lock/unlock action will be performed on storage units.
unitLockControl Array of structures, one for each storage unit to be locked or unlocked. Only valid in the case where *lockIndividual* is specified in the *cashUnitAction* property otherwise this will be ignored.
unitLockControl/storageUnit Name of the storage unit (as stated by the Storage.GetStoragecommand) to be locked or unlocked. Property value constraints: `pattern: ^unit[0-9A-Za-z]+$`
unitLockControl/unitAction Specifies whether to lock or unlock the storage unit indicated in the *storageUnit* parameter. The following values are possible: • `lock` - Locks the specified storage unit so that it cannot be removed from the device. • `unlock` - Unlocks the specified storage unit so that it can be removed from the device.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "invalidCashUnit"	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.

Properties
errorCode Specifies the error code if applicable. The following values are possible: • `invalidCashUnit` - The storage 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 storage units specified could not be locked/unlocked, e.g., the lock action could not be performed because the storage unit specified to be locked had been removed.

Event Messages

- Storage.StorageErrorEvent
- Storage.StorageThresholdEvent

8.1.18 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, CashManagement.Reset or CashAcceptor.CreateSignature command and can be used with explicit and implicit shutter control. The command is only valid on positions where *usage* reported by the CashAcceptor.GetPositionCapabilities command is *rollback* or *refuse* and where *presentControl* reported by the *CashAcceptor.GetPositionCapabilities* command is false.

This command cannot be used to present items stacked through the CashDispenser interface. Where this is attempted the command fails with errorCode *sequenceError*.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"position": "inLeft"	string	
}		

Properties

timeout

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

default: 0

position

Supplies the input or output position as one of the following values. If not specified, the default position applies. Supported positions are reported in Common.Capabilities.

- `inDefault` - Default input position.
- `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.
- `outDefault` - Default output 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.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "unsupportedPosition"	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: • `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

- CashManagement.ItemsTakenEvent
- CashManagement.ItemsPresentedEvent

8.1.19 CashAcceptor.Deplete

This command moves items from multiple storage units to a single storage unit. Applications can use this command to ensure that there are the optimum number of items in the cassettes by moving items from source storage units to a target storage unit. This is especially applicable if surplus items are removed from multiple recycle storage units to a replenishment storage unit and can help to minimize manual replenishment operations.

The CashAcceptor.GetDepleteSource command can be used to determine what storage units can be specified as source storage units for a given target storage unit.

The counts returned by the Storage.GetStorage 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 will be sent.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"depleteSources": [{	array (object)	
"source": "unit1",	string	
"numberOfItemsToMove": 100	integer	
}],		
"cashUnitTarget": "unit1"	string	
}		

Properties

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

depleteSources
Array of objects listing which storage units are to be depleted. There must be at least one element in this array.

depleteSources/source
Name of the storage unit (as stated by the *Storage.GetStorage* command) from which items are to be removed.
Property value constraints:
`pattern: ^unit[0-9A-Za-z]+$`

depleteSources/numberOfItemsToMove
The number of items to be moved from the source storage unit. If 0, all items will be moved. If non-zero, this must be equal to or less than the count of items reported for the storage unit specified by *cashUnitSource*.
Property value constraints:
`minimum: 0`

cashUnitTarget
Name of the storage unit (as stated by the *Storage.GetStorage* command) to which items are to be moved.
Property value constraints:
`pattern: ^unit[0-9A-Za-z]+$`

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	

Payload (version 1.0)	Type	Required
"errorDescription": "Device not available",	string	
"errorCode": "cashUnitError",	string	
"numberOfItemsReceived": 100,	integer	
"numberOfItemsRejected": 10,	integer	
"depleteSourceResults": [{	array (object)	
"cashUnitSource": "unit1",	string	
"cashItem": "type20USD1",	string	
"numberOfItemsRemoved": 0	integer	
}]		
}		

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:

- `cashUnitError` - A problem occurred with a storage unit. A Storage.StorageErrorEvent will be sent with the details. If appropriate a CashAcceptor.IncompleteDepleteEventwill also be sent.
- `invalidCashUnit` - The source or target storage unit specified is invalid for this operation. The CashAcceptor.GetDepleteSource 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 storage unit during execution of this command.

Property value constraints:

```
minimum: 0
```

numberOfItemsRejected

Total number of items rejected during execution of this command.

Property value constraints:

```
minimum: 0
```

depleteSourceResults

Breakdown of which notes moved where. In the case where one item type has several releases and these are moved, or where items are moved from a multi denomination storage unit to a multi denomination storage unit, each source can move several note types.

For example:

- If one single source was specified with the input structure, and this source moved two different note types, then this will have two elements.
- If two sources were specified and the first source moved two different note types and the second source moved three different note types, then this will have five elements.

depleteSourceResults/cashUnitSource

Name of the storage unit (as stated by the *Storage.GetStorage* command) from which items have been removed.

Property value constraints:

```
pattern: ^unit[0-9A-Za-z]+$
```

Properties
depleteSourceResults/cashItem A cash item as reported by CashManagement.GetBankNoteTypes. This is not specified if the item was not identified as a cash item. Property value constraints: `pattern: ^type[0-9A-Z]+$`
depleteSourceResults/numberOfItemsRemoved Total number of items removed from this source storage unit of the *cashItem* item type. Not reported if this source storage unit did not move any items of this item type, for example due to a storage unit or transport jam. Property value constraints: `minimum: 0`

Event Messages

- Storage.StorageErrorEvent
- Storage.StorageThresholdEvent
- CashManagement.NoteErrorEvent
- CashManagement.InfoAvailableEvent
- CashAcceptor.IncompleteDepleteEvent

8.1.20 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:

CashManagement.OpenShutter

CashAcceptor.PresentMedia

The application can tell whether the additional items were left by using the CashAcceptor.GetPresentStatus command. This command does not affect the status of the current cash-in transaction.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"position": "outDefault"	string	
}		

Properties

timeout

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

default: 0

position

Supplies the output position as one of the following values. Supported positions are reported in Common.Capabilities.

- `outDefault` - Default output 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.

default: "outDefault"

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "unsupportedPosition",	string	
"position": "outDefault"	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.

Properties
errorCode Specifies the error code if applicable. The 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 storage unit caused a problem. A Storage.StorageErrorEvent will be posted with the details.
position Supplies the output position as one of the following values. Supported positions are reported in Common.Capabilities. • `outDefault` - Default output 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. default: "outDefault"

Event Messages

- Storage.StorageErrorEvent
- CashManagement.InfoAvailableEvent

8.2 Event Messages

8.2.1 CashAcceptor.InputRefuseEvent

This event specifies that the device has refused either a portion or all the items.

Event Message

Payload (version 1.0)	Type	Required
`{`		
`"reason": "cashInUnitFull"`	string	
`}`		

Properties

reason

Reason for refusing a part of the amount. The following values are possible:

- `cashInUnitFull` - storage 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 storage 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 inserted exceeded the limitation set with the CashAcceptor.CashInStart command.
- `limitOverAmount` - Amount exceeded the limitation set with the *CashAcceptor.CashInStart* command.

8.2.2 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.

Event Message

Payload (version 1.0)	Type	Required
{		
"unrecognized": 0,	integer	
"*type20USD1*": {	object	
"fit": 0,	integer	
"unfit": 0,	integer	
"suspect": 0,	integer	
"counterfeit": 0,	integer	
"inked": 0	integer	
},		
"*type50USD1*": {	object	
See type20USD1 properties.		
}		
}		

Properties
unrecognized Count of unrecognized items handled by the cash interface.
type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification.
type20USD1/fit Count of genuine cash items which are fit for recycling.
type20USD1/unfit Count of genuine cash items which are unfit for recycling.
type20USD1/suspect Count of suspected counterfeit cash items.
type20USD1/counterfeit Count of counterfeit cash items.
type20USD1/inked Count of cash items which have been identified as ink stained.

8.2.3 CashAcceptor.InsertItemsEvent

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

Event Message

Payload (version 1.0)	Type	Required
{		
}		

8.2.4 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.

Event Message

Payload (version 1.0)	Type	Required
{		
"replenish": {	object	
"numberOfItemsRemoved": 20,	integer	
"numberOfItemsRejected": 2,	integer	
"replenishTargetResults": [{	array (object)	
"target": "unit1",	string	
"cashItem": "type20USD1",	string	
"numberOfItemsReceived": 20	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 storage unit including rejected items during execution of this command. Not specified if no items were removed.

Property value constraints:

```
minimum: 1
```

replenish/numberOfItemsRejected

Total number of items rejected during execution of this command. Not specified if no items were rejected.

Property value constraints:

```
minimum: 1
```

replenish/replenishTargetResults

Breakdown of which notes were moved and where they moved to. In the case where one note type has several releases and these are moved, or where items are moved from a multi denomination storage unit to a multi denomination storage unit, each target can receive several note types.

For example:

- If one single target was specified with the *replenishTargets* input structure, and this target received two different note types, then this property will have two elements.
- If two targets were specified and the first target received two different note types and the second target received three different note types, then this property will have five elements.

replenish/replenishTargetResults/target

Name of the storage unit (as stated by the Storage.GetStoragecommand) to which items have been moved.

Property value constraints:

```
pattern: ^unit[0-9A-Za-z]+$
```

Properties
replenish/replenishTargetResults/cashItem A cash item as reported by CashManagement.GetBankNoteTypes. This is not specified if the item was not identified as a cash item. Property value constraints: `pattern: ^type[0-9A-Z]+$`
replenish/replenishTargetResults/numberOfItemsReceived Total number of items received in this target storage unit of the *cashItem* note type. Property value constraints: `minimum: 1`

8.2.5 CashAcceptor.IncompleteDepleteEvent

This event is generated when 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.

Event Message

Payload (version 1.0)	Type	Required
{		
"deplete": {	object	
"numberOfItemsReceived": 100,	integer	
"numberOfItemsRejected": 10,	integer	
"depleteSourceResults": [{	array (object)	
"cashUnitSource": "unit1",	string	
"cashItem": "type20USD1",	string	
"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 storage unit during execution of this command.

Property value constraints:

```
minimum: 0
```

deplete/numberOfItemsRejected

Total number of items rejected during execution of this command.

Property value constraints:

```
minimum: 0
```

deplete/depleteSourceResults

Breakdown of which notes moved where. In the case where one item type has several releases and these are moved, or where items are moved from a multi denomination storage unit to a multi denomination storage unit, each source can move several note types.

For example:

- If one single source was specified with the input structure, and this source moved two different note types, then this will have two elements.
- If two sources were specified and the first source moved two different note types and the second source moved three different note types, then this will have five elements.

deplete/depleteSourceResults/cashUnitSource

Name of the storage unit (as stated by the *Storage.GetStorage* command) from which items have been removed.

Property value constraints:

```
pattern: ^unit[0-9A-Za-z]+$
```

deplete/depleteSourceResults/cashItem

A cash item as reported by CashManagement.GetBankNoteTypes. This is not specified if the item was not identified as a cash item.

Property value constraints:

```
pattern: ^type[0-9A-Z]+$
```

Properties
deplete/depleteSourceResults/numberOfItemsRemoved Total number of items removed from this source storage unit of the *cashItem* item type. Not reported if this source storage unit did not move any items of this item type, for example due to a storage unit or transport jam. Property value constraints: `minimum: 0`

9. Key Management Interface

This chapter defines the Key Management interface functionality and messages.

This section describes the general interface for the following functions:

- Loading of encryption keys.
- EMV 4.0 PIN blocks, EMV 4.0 public key loading, static and dynamic data verification.

Important Notes:

- This revision of this specification does not define all key management procedures; some key management is still vendor-specific.
- Key space management is customer-specific, and is therefore handled by vendor-specific mechanisms.

Key values are passed to the API as binary hexadecimal values, for example: 0123456789ABCDEF = 0x01 0x23 0x45 0x67 0x89 0xAB 0xCD 0xEF. When hex values are passed to the API within strings, the hex digits 0xA to 0xF can be represented by characters in the ranges 'a' to 'f' or 'A' to 'F'.

Certain levels of the PCI security standards specify that if a Key Encryption Key (KEK) is deleted or replaced, then all keys in the hierarchy under that KEK are also removed. When a key is deleted, clients should check the loaded state of other keys using KeyManagement.GetKeyDetail.

9.1 General Information

9.1.1 References

ID	Description
keymanagement-1	RSA Laboratories, PKCS#7: Cryptographic Message Syntax Standard. Version 1.5, November 1993.
keymanagement-2	SHA-1 Hash algorithm ANSI X9.30-2:1993, Public Key Cryptography for Financial Services Industry Part 2.
keymanagement-3	EMVCo, EMV2000 Integrated Circuit Card Specification for Payment Systems, Book 2 – Security and Key Management, Version 4.0, December 2000.
keymanagement-4	Europay International, EPI CA Module Technical – Interface specification Version 1.4.
keymanagement-5	Groupement des Cartes Bancaires "CB", Description du format et du contenu des données cryprographiques échangées entre GAB et GDG, Version 1.3 / Octobre 2002.
keymanagement-6	ANSI - X9.143, Retail Financial Services Interoperable Secure Key Block Specification.
keymanagement-7	ISO/IEC 10118-3:2004 Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions.
keymanagement-8	FIPS 180-2 Secure Hash Signature Standard.
keymanagement-9	ANS X9 TR-34 2019, Interoperable Method for Distribution of Symmetric Keys using Asymmetric Techniques: Part 1 – Using Factoring-Based Public Key Cryptography Unilateral Key Transport.
keymanagement-10	ANS X9.24-1:2009, Retail Financial Services Symmetric Key Management Part 1: Using Symmetric Techniques.
keymanagement-11	NIST Special Publication 800-38A: Recommendation for Block Cipher Modes of Operation.
keymanagement-12	NIST Special Publication 800-38E: Recommendation for Block Cipher Modes of Operation: the XTS-AES Mode for Confidentiality on Storage Devices.

9.1.2 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.

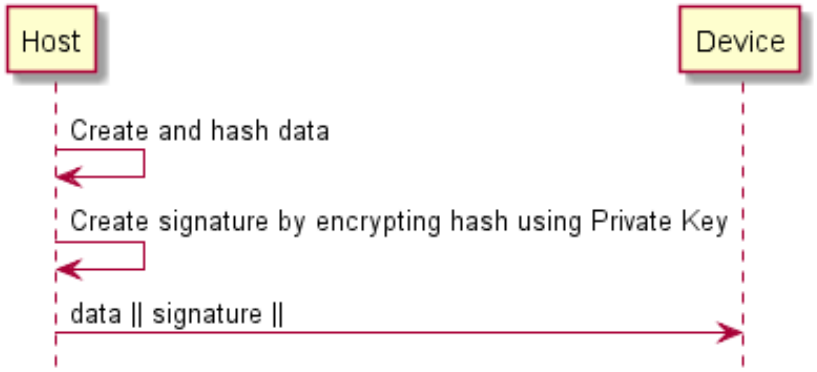
Definitions and Abbreviations	**Description**
ATM	Automated Teller Machine, used here for any type of self-service terminal, regardless whether it actually dispenses cash.
CA	Certificate Authority.
Certificate	A data structure that contains a public key and a name that allows certification of a public 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	The key of an entity's key pair that should only be used by that entity.
Public Key	The 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	Description
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.

Definitions and Abbreviations	Description
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.

9.1.3 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. The 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. The 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. The data block.
2. The digital signature.
3. The 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:

- The 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.
- The 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.
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
- 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}.

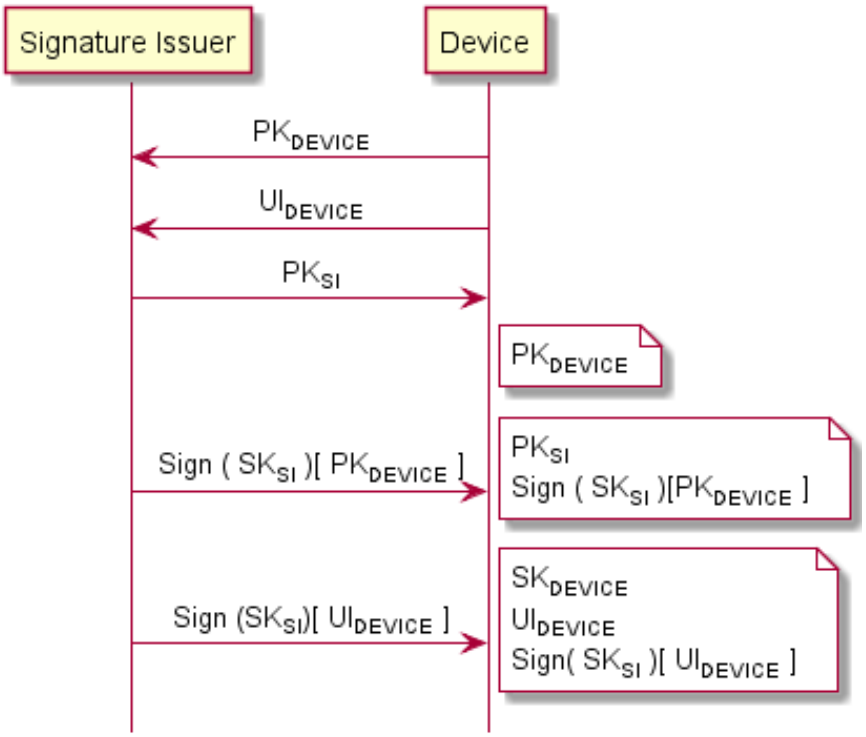
The Device will then validate the signature using PK_{HOST}, verify the random number and then obtain the master key by decrypting using SK_{DEVICE}.

The commands used to exchange master symmetric keys as described above are:

- KeyManagement.StartKeyExchage
- KeyManagement.ImportKey

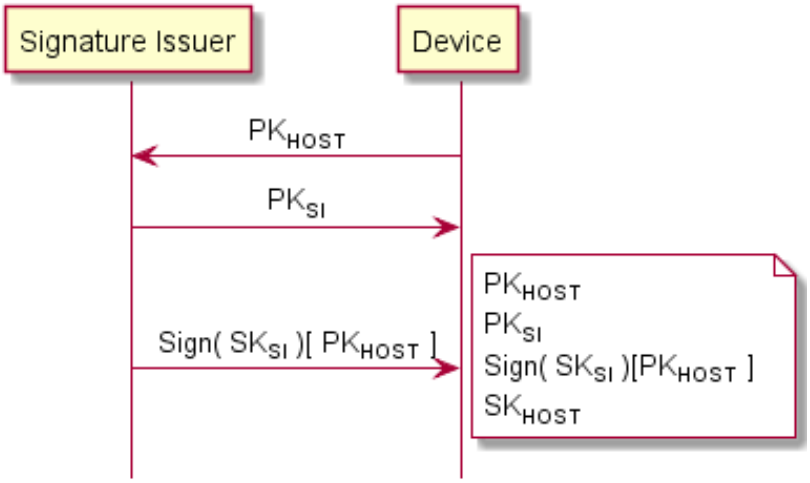
Initialization Phase – Signature Issuer and ATM PIN

This would typically occur in a secure manufacturing environment.



Initialization Phase – Signature Issuer and Host

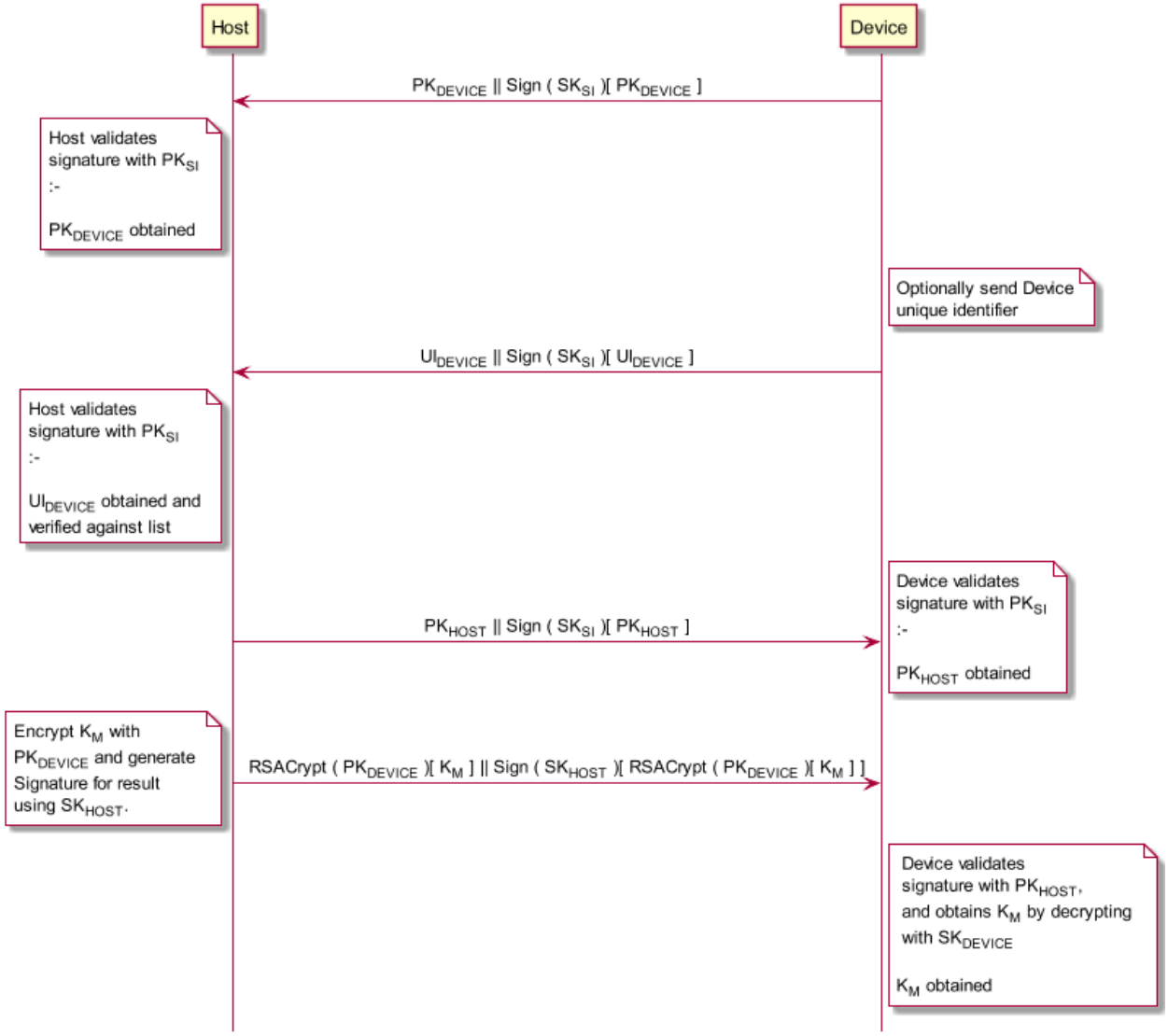
This would typically occur in a secure offline environment.



Key Exchange – Host and ATM PIN

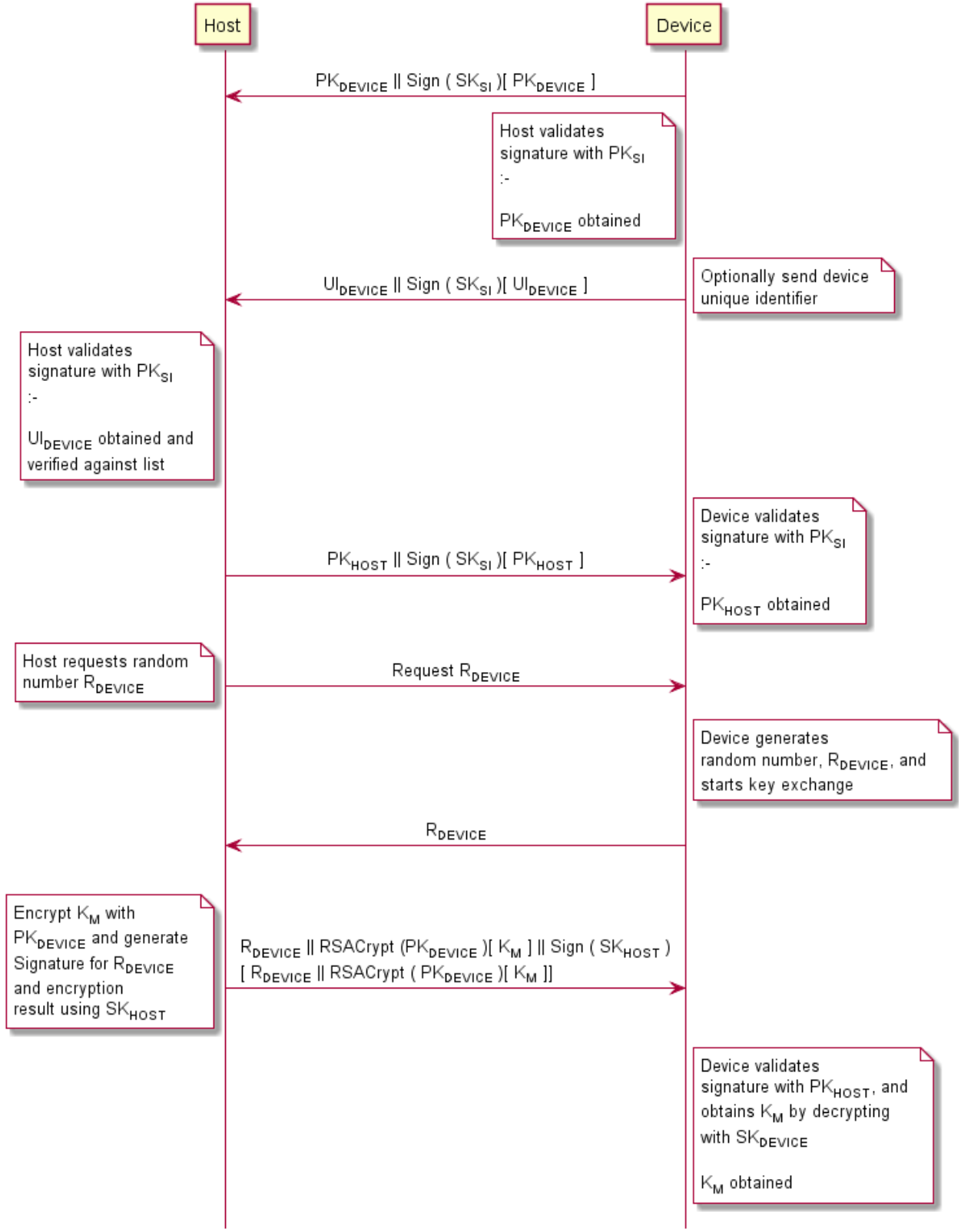
This following is a typical interaction for the exchange of the initial symmetric master key between host and device.

The following is the recommended sequence of interchanges.



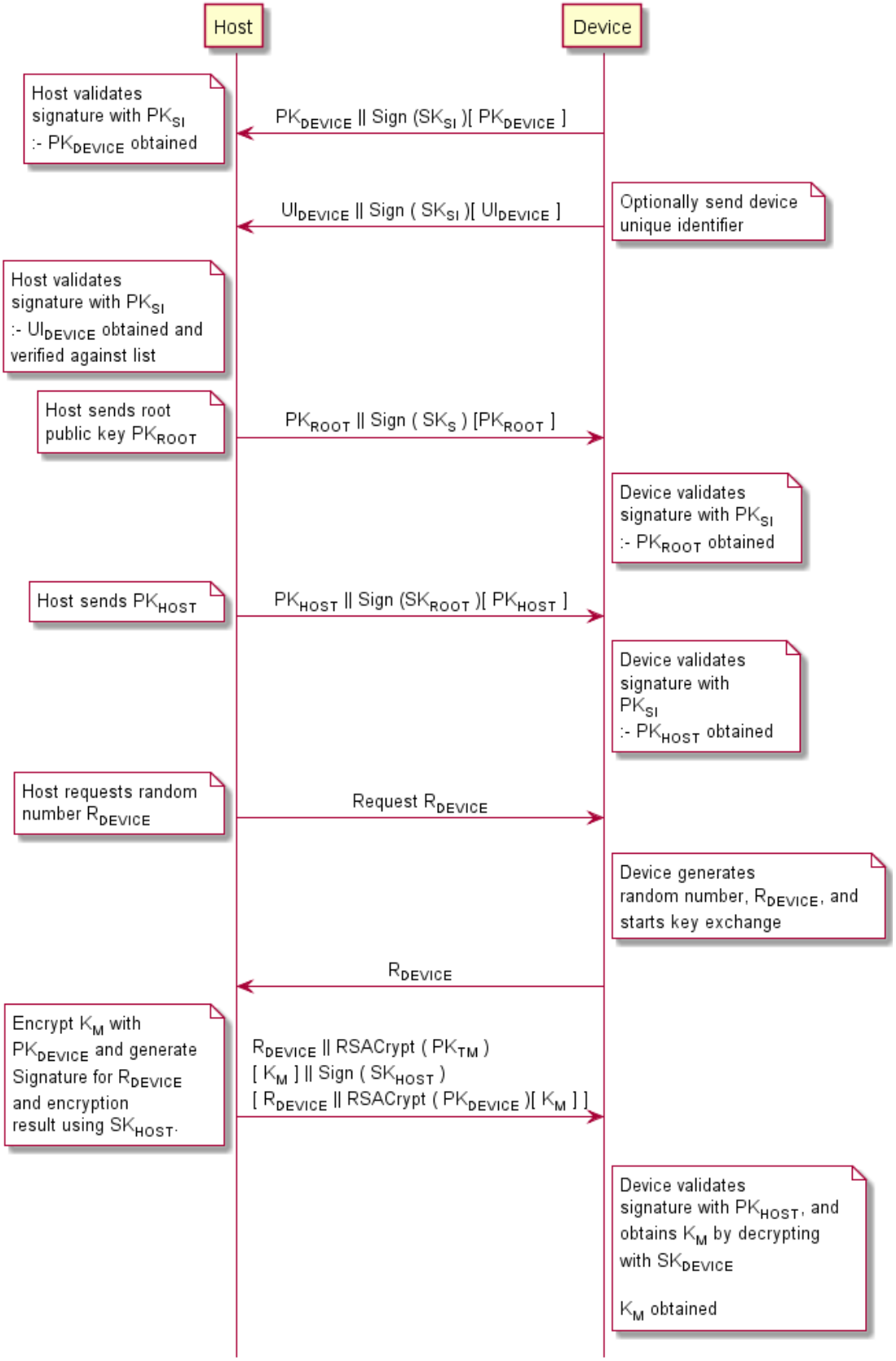
Key Exchange (with random number) – Host and ATM PIN

This following is a typical interaction for the exchange of the initial symmetric master key when the device supports the KeyManagement.StartKeyExchange command.



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.



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}
"_DeviceCryptKey"	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 Device.
"_DeviceCryptCert"	Public/Private key-pair	CA	This key is used for certificate based remote key loading when transporting symmetric 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
"_DeviceSignKey"	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.

9.1.4 Remote Key Loading Using Certificates

The following sections demonstrate the proper usage of the KeyManagement interface to accomplish Remote Key Loading using Certificates. There are sequence diagrams to demonstrate how the KeyManagement interface can be used to complete each of the TR-34 operations.

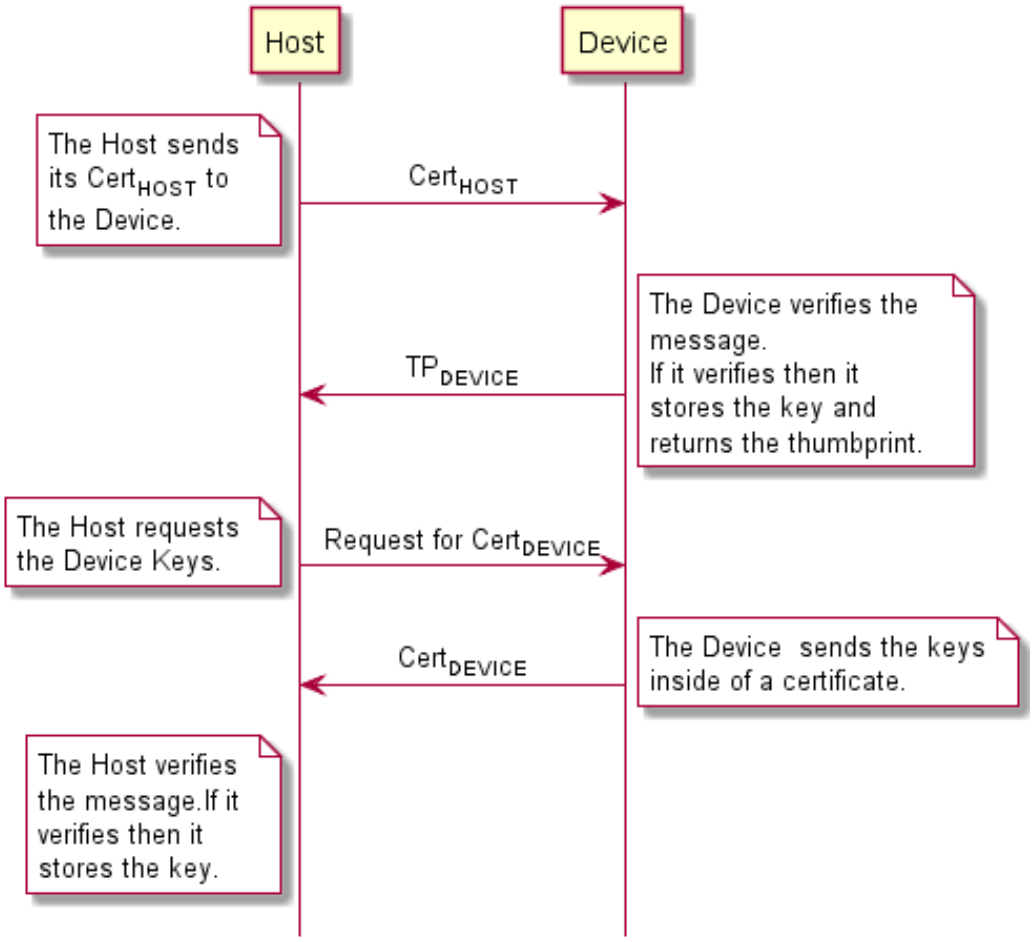
Certificate Exchange and Authentication

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:

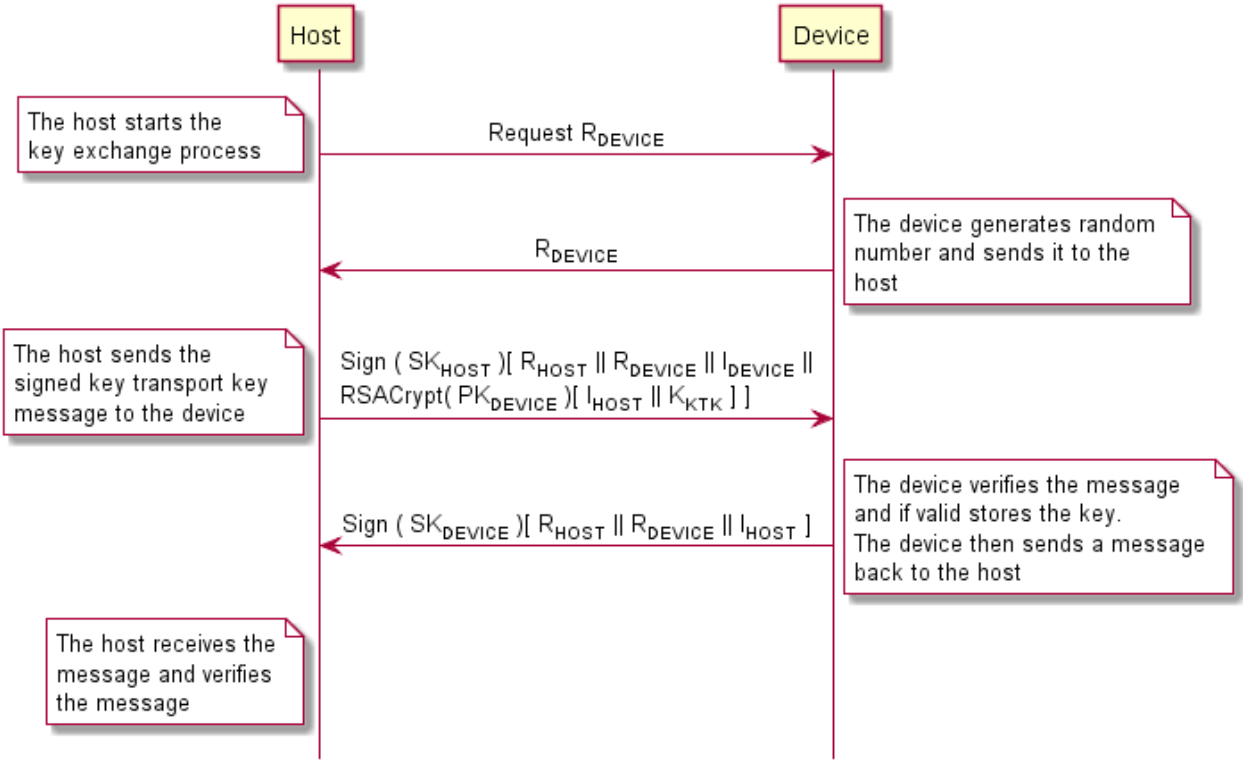


Remote Key Exchange

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.ImportKey message.
2. The Host obtains a Key Transport Key and wants to transfer it to the device. The 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.
3. 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 to the device using KeyManagement.ImportKey. 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.
4. 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.
5. 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. 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.

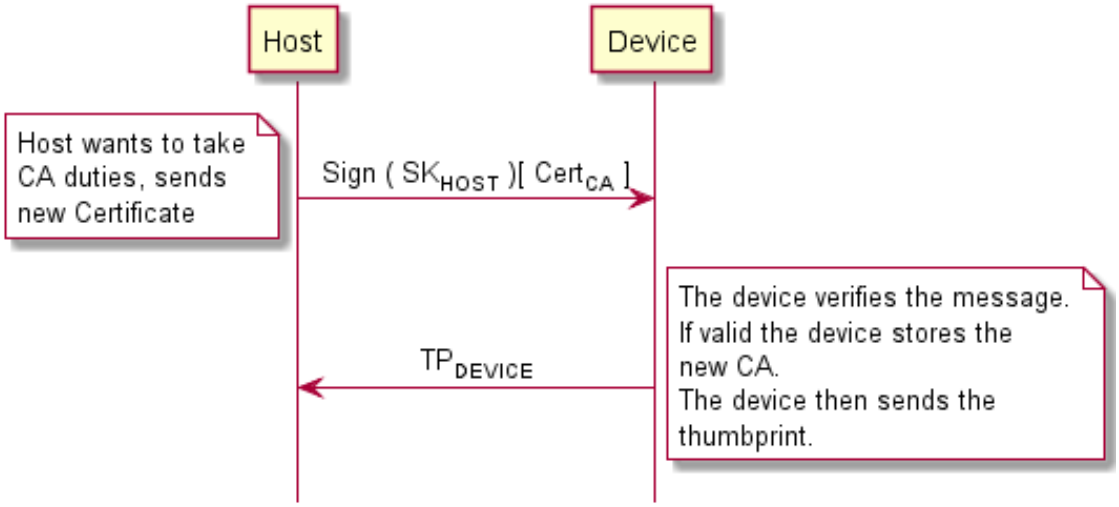
The following diagram below shows how the host and device transmit the Key Transport Key.



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 in the message. If valid, the device 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.



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 (See [Ref. keymanagement-1]) 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.

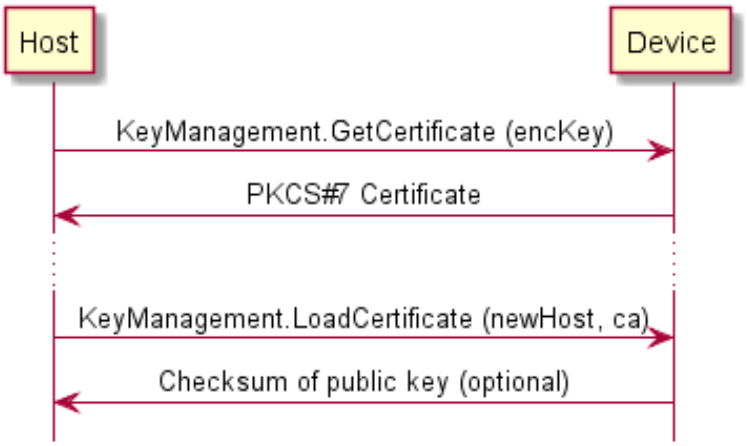
After the host tells the device to shift to the secondary certificate state, only Secondary Certificates can be used. The device will no longer be able to go back to the Primary State and any attempts from the host to get or load a Primary Certificate will return an error. When either Primary or Secondary certificates are compromised it is up to the vendor on how the device should be handled with the manufacturer.

9.1.5 Remote Key Loading Using TR34

TR34 BIND To Host

This section defines the commands to use when transferring a TR34 BIND token as defined in X9 TR34-2019 [Ref. keymanagement-9].

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.



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

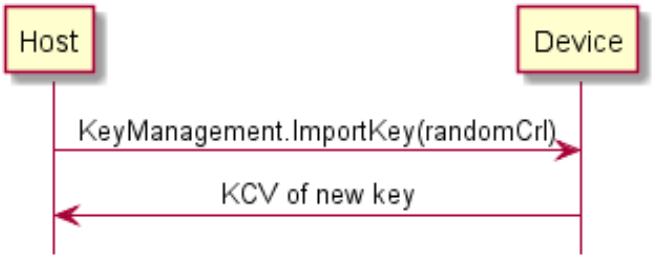
There are two protocols that can be used to transport symmetric keys under TR34; these are the One Pass and Two Pass protocols. The use of XFS4IoT commands for these two protocols are shown in the following sections.

- 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) as defined in X9 TR34-2019 [Ref. keymanagement-9].

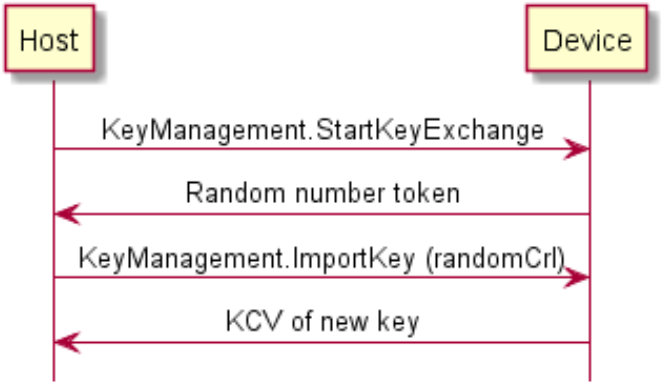
Pre-condition: A successful BIND command has completed such that the device is bound to the host.



Two Pass

This section defines the command to use when transferring a TR34 KEY token (2-pass) as defined in X9 TR34-2019 [Ref. keymanagement-9].

Pre-condition: A successful BIND command has completed such that the device is bound to the host.

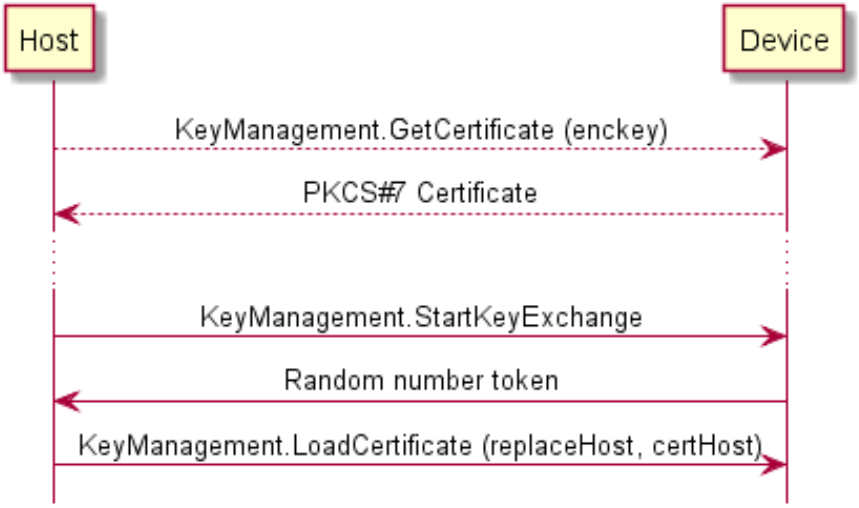


NB: Dotted lines represent commands that are only required if the device encryption certificate has not been previously stored by the host.

TR34 REBIND To New Host

This section defines the command to use when transferring a TR34 REBIND token as defined in X9 TR34-2019 [Ref. keymanagement-9].

Pre-condition: A successful BIND command has completed such that the device is bound to the host.

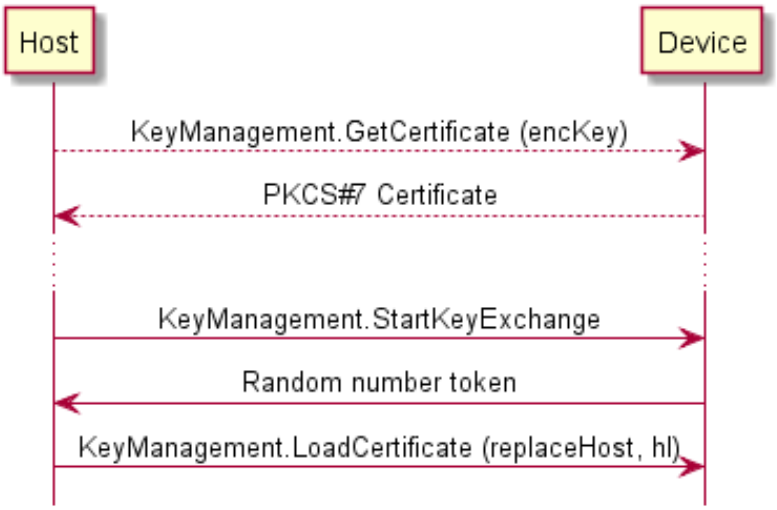


NB: Dotted lines represent commands that are only required if the device encryption certificate has not been previously stored by the host.

TR34 Force REBIND To New Host

This section defines the command to use when transferring a TR34 Force REBIND token as defined in X9 TR34-2019 [Ref. keymanagement-9].

Pre-condition: A successful BIND command has completed such that the device is bound to the host.



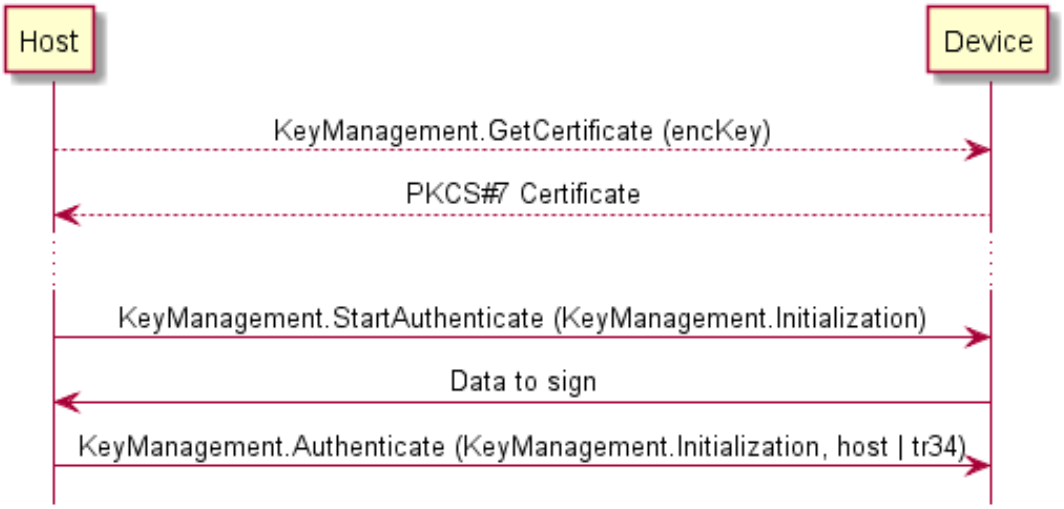
NB: Dotted lines represent commands that are only required if the device encryption certificate has not been previously stored by the host.

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 as defined in X9 TR34-2019 [Ref. keymanagement-9].

Pre-condition: A successful BIND command has completed such that the device is bound to the host.

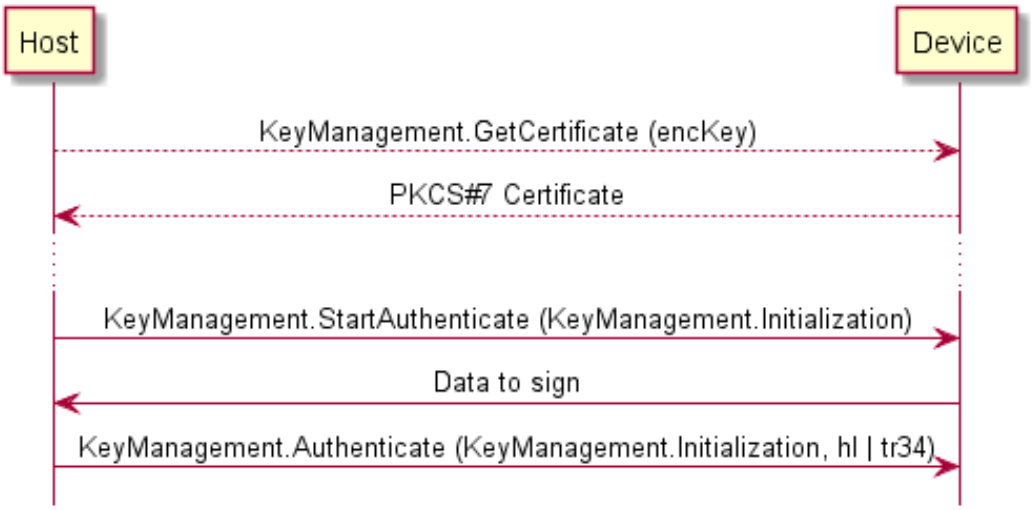


NB: Dotted lines represent commands that are only required if the device encryption certificate has not been previously stored by the host.

TR34 Force UNBIND From Host

This section defines the command to use when transferring a TR34 Force UNBIND token as defined in X9 TR34-2019 [Ref. keymanagement-9].

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.

9.1.6 EMV Support

EMV supported consists of the following:

- Import of the Certification Authority and Chip Card Public Keys
- Creating the PIN blocks for offline PIN verification and verifying static and dynamic data.

This section is used to further explain concepts and functionality that needs further clarification.

The 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 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.

Key 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 (See [Ref. keymanagement-3]).
- EPI CA (or self signed) format as specified in the Europay International, EPI CA Module Technical – Interface specification Version 1.4 (See [Ref. keymanagement-4]).
- pkcsV15 encrypted (as used by GIECB in France) (See [Ref. keymanagement-5]).

EPI CA format

The following table corresponds to table 4 of the Europay International, EPI CA Module Technical – Interface specification Version 1.4 (See [Ref. keymanagement-4]) and identifies the Europay Public Key (self-certified) and the associated data:

Field Name	Length	Description	Format
ID of Certificate Subject	5	RID for Europay	Binary

Field Name	Length	Description	Format
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.

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 (See [Ref. keymanagement-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 is generated using PinPad.GetPinBlock. The format *formEmv* is used indicate to the service that the PIN block must follow the requirements of the EMVCo, Book2 – Security & Key management Version 4.0 document. The property *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.

9.1.7 KeyManagement.ImportKey command Input-Output Parameters

This section describes the input/output parameters for various scenarios in which the KeyManagement.ImportKey command is used.

Importing a 3DES 16-byte Terminal Master Key using Signature-based Remote Key Loading

KeyManagement.ImportKey command payload

```
{
  "header": {
    "type": "command",
    "name": "KeyManagement.ImportKey",
    "requestId": 12345
  },
  "payload" : {
    "key": "testKey","
    "keyAttributes": {
      "keyUsage": "K0",
      "algorithm": "T",
      "modeOfUse": "D"
    },
    "value": "<encrypted key value>",
    "decryptKey": "deviceCryptKey",
    "decryptKey": "rsaesOaep",
    "verificationData": "<signature generated by the host>",
    "verifyKey": "hostKey",
    "verifyAttributes": {
      "cryptoMethod": "rsassaPss",
      "hashAlgorithm": "sha256"
    }
  }
}
```

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 3DES 16-byte Pin Encryption Key with a Key Check Value in the Input

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"
    }
  }
}
```

KeyManagement.ImportKey completion payload

```
{
  "header": {
    "type": "completion",
    "name": "KeyManagement.ImportKey",
    "requestId": 12345
  },
  "payload" : {
    "keyLength": 128
  }
}
```

Importing a 3DES 16-byte MAC (Algorithm 3) Key

KeyManagement.ImportKey command payload

```
{
  "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

KeyManagement.ImportKey command payload

```
{
  "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"
    }
  }
}
```

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 3DES 24-byte Data Encryption Key via an X9.143 Keyblock

KeyManagement.ImportKey command payload

```
{
  "header": {
    "type": "command",
    "name": "KeyManagement.ImportKey",
    "requestId": 12345
  },
  "payload" : {
    "key": "testKey","
    "keyAttributes": {
      "keyUsage": "D0",
      "algorithm": "T",
      "modeOfUse": "E"
    },
    "value": "<key block>",
    "decryptKey": "masterKey"
  }
}
```

KeyManagement.ImportKey completion payload

```
{
  "header": {
    "type": "completion",
    "name": "KeyManagement.ImportKey",
    "requestId": 12345
  },
  "payload" : {
    "keyLength": 192
  }
}
```

9.1.8 DUKPT

Definitions and Abbreviations	Description
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 [Ref. keymanagement-10].

The IPEK key is given a fixed name so multi-vendor applications can be developed without the need for vendor specific configuration tools.

The BDK is used to derive the IPEK. When a 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. Therefore, the IPEK will be reported as loaded.

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, to use those keys in the Crypto.CryptoData and Crypto.GenerateAuthentication commands, the IPEK key name must be used as the key name and the algorithm must be *cryptTriDesCbc* and *cryptTriDesMac* respectively.

The optional variant response data encryption and MAC keys are not supported.

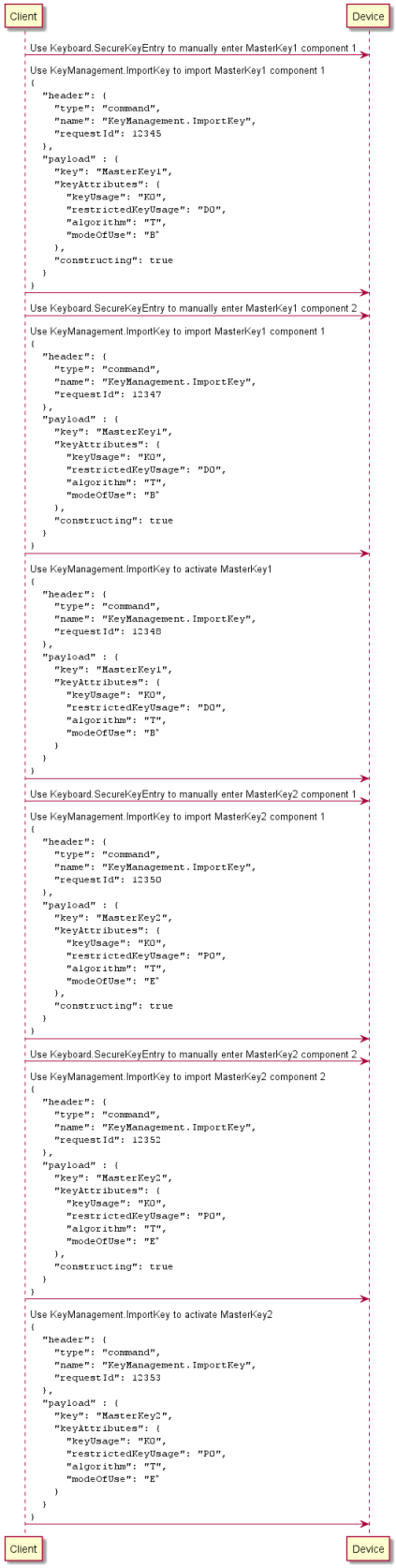
If DUKPT is supported, this key must be included in the KeyManagement.GetKeyDetail output.

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).

9.1.9 Restricted Encryption Key Command Usage

This example command flow sequence shows how encryption keys can be derived/not derived if the master key has a restricted use.

In this example the master keys are loaded using the secure key entry. Loading with RKL works in the same way.



Master key restriction prevents import of keys with incorrect usage:

9.1.10 Secure Key Entry 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.


Client
Device
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]
Client
Device

9.2 Command Messages

9.2.1 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"keyName": "Key01"	string	
}		

Properties
timeout Timeout in milliseconds for the command to complete. If set to 0, 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 (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "keyNotFound",	string	
"keyDetails": {	object	
"*exampleProperty1*": {	object	
"generation": 0,	integer	
"version": 0,	integer	
"activatingDate": "20210101",	string	
"expiryDate": "20220101",	string	
"loaded": "no",	string	Yes
"keyBlockInfo": {	object	
"keyUsage": "K0",	string	Yes
"restrictedKeyUsage": "D0",	string	
"algorithm": "T",	string	Yes
"modeOfUse": "B",	string	Yes
"keyVersionNumber": "01",	string	
"exportability": "E",	string	Yes

Payload (version 1.0)	Type	Required
"optionalBlockHeader": "00",	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
```

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])$
```

Properties
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 X9.143 keyblock header definitions.

Properties

keyDetails/exampleProperty1/keyBlockInfo/keyUsage

Specifies the intended function of the key. The following values are possible - See [Ref. keymanagement-6] :

- `B0` - BDK Base Derivation Key.
- `B1` - Initial DUKPT key.
- `B2` - Base Key Variant Key.
- `B3` - Key Derivation Key (Non ANSI X9.24).
- `C0` - CVK Card Verification Key.
- `D0` - Symmetric Key for Data Encryption.
- `D1` - Asymmetric Key for Data Encryption.
- `D2` - Data Encryption Key for Decimalization Table.
- `D3` - Data Encryption Key for Sensitive Data.
- `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).
- `E7` - EMV / Chip Asymmetric Key Pair for EMV/Smart Card based PIN/PIN Block Encryption.
- `I0` - Initialization Vector (IV).
- `K0` - Key Encryption or wrapping.
- `K1` - X9.143 Key Block Protection Key.
- `K2` - TR-34 Asymmetric Key.
- `K3` - Asymmetric Key for key agreement / key wrapping.
- `K4` - Key Block Protection Key, ISO 20038.
- `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.
- `P1` - PIN Generation Key (reserved for ANSI X9.132-202x).
- `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.
- `V5` - PIN Verification Key, ANSI X9.132 algorithm 3.
- `00 – 99` - These numeric values are reserved for proprietary use.

Property value constraints:

```
pattern: ^B[0-3]$|^C0$|^D[0-3]$|^E[0-7]$|^I0$|^K[0-4]$|^M[0-8]$|^P[0-1]$|^S[0-2]$|^V[0-5]$|^[0-9][0-9]$
```

Properties

keyDetails/exampleProperty1/keyBlockInfo/restrictedKeyUsage

If the *keyUsage* is a key encryption usage (e.g. 'K0') this specifies the key usage of the keys that can be encrypted by the key.

The following values are possible:

- `B0` - BDK Base Derivation Key.
- `B1` - Initial DUKPT key.
- `B2` - Base Key Variant Key.
- `B3` - Key Derivation Key (Non ANSI X9.24).
- `C0` - CVK Card Verification Key.
- `D0` - Symmetric Key for Data Encryption.
- `D1` - Asymmetric Key for Data Encryption.
- `D2` - Data Encryption Key for Decimalization Table.
- `D3` - Data Encryption Key for Sensitive Data.
- `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).
- `E7` - EMV / Chip Asymmetric Key Pair for EMV/Smart Card based PIN/PIN Block Encryption.
- `I0` - Initialization Vector (IV).
- `K0` - Key Encryption or wrapping.
- `K1` - X9.143 Key Block Protection Key.
- `K2` - TR-34 Asymmetric Key.
- `K3` - Asymmetric Key for key agreement / key wrapping.
- `K4` - Key Block Protection Key, ISO 20038.
- `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.
- `P1` - PIN Generation Key (reserved for ANSI X9.132-202x).
- `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.
- `V5` - PIN Verification Key, ANSI X9.132 algorithm 3.
- `00 – 99` - These numeric values are reserved for proprietary use.

This should be omitted if the key usage is not an key encryption usage or restricted key encryption keys are not supported or required.

Property value constraints:

```
pattern: ^B[0-3]$|^C0$|^D[0-3]$|^E[0-7]$|^I0$|^K[0-4]$|^M[0-8]$|^P[0-1]$|^S[0-2]$|^V[0-5]$|^[0-9][0-9]$
```

Properties
keyDetails/exampleProperty1/keyBlockInfo/algorithm Specifies the algorithm for which the key can be used. See [Ref. keymanagement-6] for all possible values: • 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]$`
keyDetails/exampleProperty1/keyBlockInfo/modeOfUse Specifies the operation that the key can perform. See [Ref. keymanagement-6] for all possible values: • B - Both Encrypt and Decrypt / Wrap and unwrap. • C - Both Generate and Verify. • D - Decrypt / Unwrap Only. • E - Encrypt / Wrap Only. • G - Generate Only. • N - No special restrictions. • S - Signature Only. • T - Both Sign and Decrypt. • V - Verify Only. • X - Key used to derive other keys(s). • Y - Key used to create key variants. • 0 – 9 - These numeric values are reserved for proprietary use. Property value constraints: `pattern: ^[0-9BCDEGNSTVXY]$`
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 [Ref. keymanagement-6] 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]$`
keyDetails/exampleProperty1/keyBlockInfo/exportability Specifies whether the key may be transferred outside of the cryptographic domain in which the key is found. See [Ref. keymanagement-6] for all possible values: • E - Exportable under a KEK in a form meeting the requirements of X9.24 Parts 1 or 2. • N - Non-exportable by the receiver of the key block, or from storage. Does not preclude exporting keys derived from a non-exportable key. • S - Sensitive, Exportable under a KEK in a form not necessarily meeting the requirements of X9.24 Parts 1 or 2. • 0 – 9 - These numeric values are reserved for proprietary use. Property value constraints: `pattern: ^[0-9ESN]$`
keyDetails/exampleProperty1/keyBlockInfo/optionalBlockHeader Contains any optional header blocks, as defined in [Ref. keymanagement-6]. This value can be omitted if there are no optional block headers.

Properties
keyDetails/exampleProperty1/keyBlockInfo/keyLength Specifies the length, in bits, of the key. 0 if the key length is unknown. Property value constraints: `minimum: 0`

Event Messages

None

9.2.2 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 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 (version 1.0)	Type	Required

{
  "timeout": 5000,
  "authentication": {
  "method": "none",
  "key": "Key01",
  "data": "QXV0aGVudGljYXRpb24g ..."
  }
}
	integer
	object
	string
	string
	string	

Properties

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

Properties
authentication This property can be used to include authentication data if required by the command. Additionally, if the command requires authentication: • The KeyManagement.StartAuthenticate command must be called before this command to. • Commands which do not clear or modify the authentication data from the device may be executed between the *KeyManagement.StartAuthenticate* and *keyManagement.Authenticate* command requests. • If prior to this command request, *KeyManagement.StartAuthenticate* is not called or a command clears the authentication data from the device, sequenceError will be returned.
authentication/method Specifies the method used to generate the authentication data. The possible values are: • `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 *method* is cbcmac or mac, then this is the name of a key which has a MAC key usage e.g. M0. If *method* is sigHost, then this specifies the name of a previously loaded asymmetric key (i.e. an RSA Public Key). If this contains the name of the default Signature Issuer or if omitted, the default Signature Issuer public key (installed in a secure environment during manufacture) will be used.

authentication/data
This property contains the authenticated data (MAC, Signature) generated from the previous call to either the KeyManagement device during the previous call to KeyManagement.StartAuthenticate.
The authentication method specified by *method* is used to generate this data. Both this authentication data and the data used to generate the authentication data must be verified before the operation is performed.
If *certHost*, *ca*, or *hl* is specified in the *method* property, this contains a PKCS#7 signedData structure which includes the data that was returned by KeyManagement.StartAuthenticate. The optional CRL field may or may not be included in the PKCS#7 signedData structure.
If *certHostTr34*, *caTr34* or *hlTr34* is specified in the *method* property, please refer to the X9 TR34-2019 [Ref. keymanagement-9] for more details.
If *sigHost* is specified in the *method* property, this is a PKCS#7 structure which includes the data that was returned by the *KeyManagement.StartAuthenticate* command.
If *cmcmac* or *cmac* is specified in the *method* property, then *key* must refer to a key with a MAC key usage key e.g. M0.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	string	
`"errorCode": "accessDenied"`	string	

Payload (version 1.0)	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.
- `randomInvalid` - The encrypted random number in the input data does not match the one previously provided by the device.

Event Messages

None

9.2.3 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"derivationAlgorithm": "chipZka",	string	Yes
"key": "Key01",	string	Yes
"keyGenKey": "Key02",	string	Yes
"ivKey": "IVKey01",	string	
"iv": "REVTIGluaXRpYWxpemF0 ...",	string	
"padding": 0,	integer	
"inputData": "a2V5IGRlcml2YXRpb24g ..."	string	Yes
}		

Properties

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

derivationAlgorithm
Specifies the algorithm that is used for derivation. See derivationAlgorithms) for the supported valued.

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.

ivKey
Specifies the name of the stored key used to decrypt the *iv* to obtain the Initialization Vector.
If this field is omitted, *iv* is used as the Initialization Vector.

iv
DES initialization vector for the encryption step within the derivation.
Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

padding
Specifies the padding character for the encryption step within the derivation.
Property value constraints:

```
minimum: 0
maximum: 255
```

inputData
Data to be used for key derivation.
Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	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.
- `keyNotFound` - The specified *keyGenKey* was not found.
- `keyNoValue` - The specified *keyGenKey* is not loaded.
- `algorithmNotSupported` - The specified *derivationAlgorithm* is not supported.
- `duplicateKey` - A *key* exists with that name and cannot be overwritten.
- `useViolation` - The specified *keyGenKey* usage does not support key derivation.
- `invalidKeyLength` - The length of *iv* is not supported or the length of an encryption key is not compatible with the encryption operation required.

Event Messages

None

9.2.4 KeyManagement.Reset

Sends a service reset to the Service.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

Properties

completionCode
The completion code.

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

Event Messages

None

9.2.5 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"key": "Key01",	string	Yes
"keyAttributes": {	object	
"keyUsage": "P0",	string	
"algorithm": "T",	string	
"modeOfUse": "G",	string	
"restrictedKeyUsage": "K0"	string	
},		
"value": "a2V5IHZhbHVl",	string	Yes
"constructing": false,	boolean	
"decryptKey": "Key01",	string	
"decryptMethod": "ecb",	string	
"verificationData": "ZGF0YSB0byBiZSB2ZXJp ...",	string	
"verifyKey": "VerifyKey01",	string	
"verifyAttributes": {	object	
"cryptoMethod": "kcvNone",	string	
"hashAlgorithm": "sha1"	string	
},		
"vendorAttributes": "See vendor documentation"	string	
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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.

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.
- `B3` - Key Derivation Key (Non ANSI X9.24).
- `C0` - CVK Card Verification Key.
- `D0` - Symmetric Key for Data Encryption.
- `D1` - Asymmetric Key for Data Encryption.
- `D2` - Data Encryption Key for Decimalization Table.
- `D3` - Data Encryption Key for Sensitive Data.
- `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).
- `E7` - EMV / Chip Asymmetric Key Pair for EMV/Smart Card based PIN/PIN Block Encryption.
- `I0` - Initialization Vector (IV).
- `K0` - Key Encryption or wrapping.
- `K1` - X9.143 Key Block Protection Key.
- `K2` - TR-34 Asymmetric Key.
- `K3` - Asymmetric Key for key agreement / key wrapping.
- `K4` - Key Block Protection Key, ISO 20038.
- `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.
- `P1` - PIN Generation Key (reserved for ANSI X9.132-202x).
- `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.
- `V5` - PIN Verification Key, ANSI X9.132 algorithm 3.
- `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$|^K[0-4]$|^M[0-8]$|^P0$|^S[0-2]$|^V[0-4]$|^[0-9][0-9]$
```

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

Properties

keyAttributes/restrictedKeyUsage

This property should only be included if the keyUsage is an key encryption key usage (K* e.g. 'K0') 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.
- `B3` - Key Derivation Key (Non ANSI X9.24).
- `C0` - CVK Card Verification Key.
- `D0` - Symmetric Key for Data Encryption.
- `D1` - Asymmetric Key for Data Encryption.
- `D2` - Data Encryption Key for Decimalization Table.
- `D3` - Data Encryption Key for Sensitive Data.
- `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).
- `E7` - EMV / Chip Asymmetric Key Pair for EMV/Smart Card based PIN/PIN Block Encryption.
- `I0` - Initialization Vector (IV).
- `K0` - Key Encryption or wrapping.
- `K1` - X9.143 Key Block Protection Key.
- `K2` - TR-34 Asymmetric Key.
- `K3` - Asymmetric Key for key agreement / key wrapping.
- `K4` - Key Block Protection Key, ISO 20038.
- `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.
- `P1` - PIN Generation Key (reserved for ANSI X9.132-202x).
- `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.
- `V5` - PIN Verification Key, ANSI X9.132 algorithm 3.
- `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$|^K[0-4]$|^M[0-8]$|^P0$|^S[0-2]$|^V[0-4]$|^[0-9][0-9]$
```

Properties

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.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

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 X9.143 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 (See [Ref. keymanagement-11]). • `xts` - The XTS method defined in NIST SP800-38E (See [Ref. keymanagement-12]). 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. X9.143 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.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

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.

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'), this property can be omitted. If the verifyKey algorithm is 'A', 'D', or 'T' and specified verifyKey is key usage '00', 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: • `sha1` - The SHA 1 digest algorithm. • `sha256` - The SHA 256 digest algorithm, as defined in ISO/IEC 10118-3:2004 [Ref. keymanagement-7] and FIPS 180-2 [Ref. keymanagement-8]. If the specified verifyKey is key usage any of the MAC usages (i.e. 'M1'), then this property can be omitted.
vendorAttributes Specifies the vendor attributes of the key to be imported. Refer to vendor documentation for details.

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	string	
`"errorCode": "keyNotFound",`	string	
`"verificationData": "dmVyaWZpY2F0aW9uIGRh ...",`	string	
`"verifyAttributes": {`	object	
`"keyUsage": "M0",`	string	
`"algorithm": "T",`	string	
`"modeOfUse": "V",`	string	
`"cryptoMethod": "kcvNone",`	string	
`"hashAlgorithm": "sha1"`	string	
`},`		
`"keyLength": 0`	integer	
`}`		

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

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` - 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 device. • `signatureInvalid` - The verification data in the input data is invalid. The key is not stored in the device. • `randomInvalid` - The encrypted random number in the input data does not match the one previously provided by the device. The key is not stored in the device. • `algorithmNotSupported` - The algorithm specified by *algorithm* 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 property can be omitted if there is no verification data.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

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 properties. This property should be omitted if there is no verification data.
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]$`

Properties

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]$
```

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'), this property can be omitted.

If the *algorithm* property is 'A', 'D', or 'T' and specified *keyUsage* property is '00', 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', 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:

- `sha1` - The SHA 1 digest algorithm.
- `sha256` - The SHA 256 digest algorithm, as defined in ISO/IEC 10118-3:2004 [Ref. keymanagement-7] and FIPS 180-2 [Ref. keymanagement-8].

If the *keyUsage* property is any of the MAC usages (e.g. 'M1'), this property can be omitted.

keyLength

Specifies the length, in bits, of the key. Zero if the key length is unknown.

Property value constraints:

```
minimum: 0
```

Event Messages

None

9.2.6 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"authentication": {	object	
"method": "none",	string	
"key": "Key01",	string	
"data": "QXV0aGVudGljYXRpb24g ..."	string	
},		
"key": "Key01"	string	
}		

Properties

timeout

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

default: 0

authentication

This property can be used to include authentication data if required by the command.

Additionally, if the command requires authentication:

- The KeyManagement.StartAuthenticate command must be called before this command to.
- Commands which do not clear or modify the authentication data from the device may be executed between the *KeyManagement.StartAuthenticate* and *keyManagement.Authenticate* command requests.
- If prior to this command request, *KeyManagement.StartAuthenticate* is not called or a command clears the authentication data from the device, sequenceError will be returned.

authentication/method

Specifies the method used to generate the authentication data. The possible values are:

- `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 *method* is cbcmac or mac, then this is the name of a key which has a MAC key usage e.g. M0.

If *method* is sigHost, then this specifies the name of a previously loaded asymmetric key (i.e. an RSA Public Key).

If this contains the name of the default Signature Issuer or if omitted, the default Signature Issuer public key (installed in a secure environment during manufacture) will be used.

Properties

authentication/data
This property contains the authenticated data (MAC, Signature) generated from the previous call to either the KeyManagement device during the previous call to KeyManagement.StartAuthenticate.
The authentication method specified by *method* is used to generate this data. Both this authentication data and the data used to generate the authentication data must be verified before the operation is performed.
If *certHost*, *ca*, or *hl* is specified in the *method* property, this contains a PKCS#7 signedData structure which includes the data that was returned by KeyManagement.StartAuthenticate. The optional CRL field may or may not be included in the PKCS#7 signedData structure.
If *certHostTr34*, *caTr34* or *hlTr34* is specified in the *method* property, please refer to the X9 TR34-2019 [Ref. keymanagement-9] for more details.
If *sigHost* is specified in the *method* property, this is a PKCS#7 structure which includes the data that was returned by the *KeyManagement.StartAuthenticate* command.
If *cmcmac* or *cmac* is specified in the *method* property, then *key* must refer to a key with a MAC key usage key e.g. M0.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

key Specifies the name of key being deleted. if this property is omitted. all keys are deleted.

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	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. • `keyNotFound` - The specified key name is not found. • `randomInvalid` - The encrypted random number in the input data does not match the one previously provided by the device. The key can not be deleted.

Event Messages

None

9.2.7 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.

See section Default Keys and Security Item loaded during manufacture for the default names and the description of the keys installed during manufacture. These names are defined to ensure multi-vendor applications can be developed.

The KeyManagement.GetKeyDetail command can be used to determine the valid uses for the exported public key.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"exportItemType": "deviceId",	string	
"name": "PKey01"	string	
}		

Properties

timeout
Timeout in milliseconds for the command to complete. If set to 0, 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 Default Keys and Security Item loaded during manufacturefor a definition of these default keys.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "noRSAKeyPair",	string	
"value": "aXRlbSBkYXRhIHJlcXVl ...",	string	
"rsaSignatureAlgorithm": "na",	string	
"signature": "U2lnbmF0dXJlIGRhdGE="	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 device's Security Item, which may be vendor specific.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

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
The RSA signature of the data item exported.
This should be omitted when the key signature is not supported.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

Event Messages

None

9.2.8 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.ExportRSADeviceSignedItem. 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"key": "Key02",	string	
"use": "rsaPrivate",	string	
"modulusLength": 0,	integer	
"exponentValue": "default"	string	
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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.GetKeyDetail 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.

Property value constraints:

```
minimum: 0
```

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).

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	

Payload (version 1.0)	Type	Required
"errorDescription": "Device not available",	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 device is unable to generate a key pair.

Event Messages

None

9.2.9 KeyManagement.ExportRSADeviceSignedItem

This command is used to export data elements from the device that have been signed by a private key within the device. 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"exportItemType": "deviceId",	string	
"name": "PKey01",	string	
"sigKey": "SigKey01",	string	
"signatureAlgorithm": "na"	string	
}		

Properties

timeout

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

default: 0

exportItemType

Defines the type of data item to be exported from the device. The possible values are:

- `deviceId` - The Unique ID for the device will be exported.
- `publicKey` - The public key identified by name will be exported.

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 (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "noRSAKeyPair",	string	
"value": "aXRlbSBkYXRhIHJlcXVl ...",	string	
"selfSignature": "c2lnbmF0dXJlIG9mIHRo ...",	string	

Payload (version 1.0)	Type	Required
"signature": "c2lnbmF0dXJlIG9mIHRo ..."	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 device's Security Item, which may be vendor specific.

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

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.

This should be omitted if not supported/required.

Property value constraints:

```
format: base64
```

signature

Specifies the RSA signature of the data item exported.

This should be omitted if not supported/required.

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

Event Messages

None

9.2.10 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 (See [Ref. keymanagement-1]) message the resulting Primary or Secondary certificate.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"getCertificate": "enckey"	string	Yes
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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 (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "accessDenied",	string	
"certificate": "Y2VydGlmaWNhdGUgREVS ..."	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.
- `invalidCertificateState` - The certificate module is in a state in which the request is invalid.
- `keyNotFound` - The specified public key was not found.

Properties

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 (See [Ref. keymanagement-1]) 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.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

Event Messages

None

9.2.11 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"replaceCertificate": "UEtDUyAjNyBkYXRh"	string	Yes
}		

Properties

timeout

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

default: 0

replaceCertificate

The PKCS#7 (See [Ref. keymanagement-1]) message that will replace the current Certificate Authority. The outer content uses the Signed-data 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.

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "accessDenied",	string	
"newCertificateData": "UEtDUyAjNyB0aHVtYiBw ..."	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.

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 PKCS#7 (See [Ref. keymanagement-1]) using a Digested-data content type. The digest parameter should contain the thumb print value.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

Event Messages

None

9.2.12 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 (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

timeout

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

default: 0

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "accessDenied",	string	
"randomItem": "Tm9uY2U="	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.

randomItem

The randomly generated number created by the device.

This is omitted if the device does not support random number generation for data authentication.

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

Event Messages

None

9.2.13 KeyManagement.GenerateKCV

This command returns the Key Check Value (KCV) for the specified key.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"key": "Key01",	string	Yes
"keyCheckMode": "self"	string	Yes
}		

Properties

timeout
Timeout in milliseconds for the command to complete. If set to 0, 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 *self* 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 (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "keyNotFound",	string	
"kcv": "a2N2"	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 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.

Properties

kcv
Contains KCV data that can be used for verification of the key.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

Event Messages

None

9.2.14 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"loadOption": "newHost",	string	Yes
"signer": "certHost",	string	Yes
"certificateData": "Y2VydGlmaWNhdGUgaW4g ..."	string	Yes
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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 (See [Ref. keymanagement-1]) using the 'degenerate certificate only' case of the SignedData 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 (See [Ref. keymanagement-1]) using the degenerate certificate.

The optional CRL field may or may not be included in the PKCS#7 (See [Ref. keymanagement-1]) signed-data structure.

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	

Payload (version 1.0)	Type	Required
"errorDescription": "Device not available",	string	
"errorCode": "accessDenied",	string	
"rsaKeyCheckMode": "none",	string	
"rsaData": "UEtDUyAjNyBkYXRh"	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.
- `signatureInvalid` - The verification data in the input data is invalid.
- `randomInvalid` - The encrypted random number in the input data does not match the one previously provided by the device.
- `modeNotSupported` - The *loadOption* and *signer* are not supported.

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 PKCS#7 (See [Ref. keymanagement-1]) 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, this property is omitted.

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

Event Messages

None

9.2.15 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 authorizationRequired.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"command": {	object	Yes
"deleteKey": {	object	
"key": "Key01"	string	
},		
"initialization": {	object	
}		
}		
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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 (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"dataToSign": "QXV0aGVudGljYXRpb24g ...",	string	
"signers": "none"	string	

Payload (version 1.0)	Type	Required
}		

Properties

completionCode

The completion code.

errorDescription

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

dataToSign

The data that must be authenticated by one of the authorities indicated by *methods* before the *command* can be executed. If the *command* does not require authentication, this property is omitted and the command result is success.

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

signers

Specifies the method used to generate the authentication data. The possible values are:

- `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

9.3 Event Messages

9.3.1 KeyManagement.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.

Event Message

Payload (version 1.0)	Type	Required
{		
"key": "Key01",	string	Yes
"ksn": "S2V5IFNlcmlhbCBOdW1i ..."	string	Yes
}		

Properties

key
Specifies the name of the DUKPT Key derivation key.

ksn
The KSN.
Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

9.4 Unsolicited Messages

9.4.1 KeyManagement.InitializedEvent

This is generated when a KeyManagement.Initialization command completed successfully.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
}		

9.4.2 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.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
"keyName": "Key02",	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.
- `algorithmNotSupported` - The specified algorithm is not supported by this key.
- `dukptOverflow` - The DUKPT KSN encryption counter has overflowed to zero. A new IPEK must be loaded.

9.4.3 KeyManagement.CertificateChangeEvent

This event indicates that the certificate module state has changed from Primary to Secondary.

Unsolicited Message

Payload (version 1.0)	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.

10. Crypto Interface

This chapter defines the Crypto interface functionality and messages.

10.1 General Information

10.1.1 References

ID	Description
crypto-1	ISO/IEC 10118-3:2004 Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions
crypto-2	FIPS 180-2 Secure Hash Signature Standard
crypto-3	ANSI X9.24-1:2009, Retail Financial Services Symmetric Key Management Part 1: Using Symmetric Techniques
crypto-4	NIST Special Publication 800-38A: Recommendation for Block Cipher Modes of Operation
crypto-5	NIST Special Publication 800-38E: Recommendation for Block Cipher Modes of Operation: the XTS-AES Mode for Confidentiality on Storage Devices

10.2 Command Messages

10.2.1 Crypto.GenerateRandom

This command is used to generate a random number.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "accessDenied",	string	
"randomNumber": "VGhlIGdlbmVyYXRlZCBy ..."	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.

randomNumber
The random number.
Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

Event Messages

None

10.2.2 Crypto.CryptoData

This command is used to encrypt or decrypt data. The *key* Mode of Use and optional *modeOfUse* property determines whether encryption or decryption will be performed.

If padding is required, the service will add it using the *padding* parameter. Clients can use an alternative padding method by pre-formatting the data and combining this with the standard padding method.

For symmetric key encryption using the CBC or CFB *cryptoMethod*, the Initialization Vector (*iv*) can be provided as input to this command, or a pre-imported IV referenced by name can be used. The *ivKey* and *iv* are both optional properties.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"key": "Key001",	string	Yes
"ivKey": "IVKey",	string	
"iv": "VGhlIGluaXRpYWxpemF0 ...",	string	
"padding": 255,	integer	
"modeOfUse": "E",	string	
"cryptoMethod": "ecb",	string	
"data": "U2FtcGxlIERhdGE="	string	Yes
}		

Properties

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

key
Specifies the name of the encryption key. The key usage must one of the supported *cryptoAttributes*.

ivKey
The name of a key used to decrypt the *iv* or the name of an Initialization Vector (IV).
If *iv* is included, this specifies the name of a key (usage 'K0') used to decrypt the *iv*.
If *iv* is omitted, this specifies the name of an IV (usage 'I0').
This is only used when the *key* usage is a symmetric encryption key and *cryptoMethod* is either CBC or CFB.

iv
The plaintext or encrypted IV for use with the CBC or CFB encryption methods.
If *iv* and *ivKey* properties are omitted the default IV is all zeroes.
Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

padding
Specifies the padding character to use for symmetric key encryption.
Property value constraints:

```
minimum: 0
maximum: 255
```

Properties
modeOfUse The *key* Mode of Use qualifier. If the *key* Mode Of Use is 'B', this qualifies the Mode of Use as one of the following values: • `D` - Decrypt / Unwrap Only. • `E` - Encrypt / Wrap Only. If the *key* Mode of Use is not 'B', this should be omitted. Property value constraints: `pattern: ^[DE]$`
cryptoMethod Specifies the cryptographic method to use. If the *key* usage 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. If the *key* usage is 'D1', this can be one of the following values: • `rsaesPkcs1V15` - Use the RSAES_PKCS1-v1.5 algorithm. • `rsaesOaep` - Use the RSAES OAEP algorithm.

data
The data to be encrypted or decrypted.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	string	
`"errorCode": "accessDenied",`	string	
`"data": "U2FtcGxlIERhdGE="`	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.

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. • `keyNotFound` - The *key* name does not exist. • `keyNoValue` - The *key* name exists but the key is not loaded. • `useViolation` - The *key* usage is not supported. • `modeOfUseNotSupported` - The *key* Mode of Use or the *modeOfUse* qualifier is not supported. • `invalidKeyLength` - The length of *iv* is not supported or the length of an encryption key is not compatible with the encryption operation required. • `cryptoMethodNotSupported` - The cryptographic method specified by *cryptoMethod* is not supported. • `noChipTransactionActive` - A chipcard key is used as encryption key and there is no chip transaction active.

data
The encrypted or decrypted data.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

Event Messages

- KeyManagement.DUKPTKSNEvent

10.2.3 Crypto.GenerateAuthentication

This command is used to generate a Message Authentication Code (MAC) or Signature.

If padding is required, the service will add it using the *padding* parameter. Clients can use an alternative padding method by pre-formatting the data and combining this with the standard padding method.

For MAC generation using the CBC or CFB *cryptoMethod*, the Initialization Vector (*iv*) can be provided as input to this command, or a pre-imported IV referenced by name can be used. The *ivKey* and *iv* are both optional properties.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"key": "Key001",	string	Yes
"data": "VGhlIEJhc2U2NCBlbmNv ...",	string	Yes
"ivKey": "IVKey",	string	
"iv": "VGhlIGluaXRpYWxpemF0 ...",	string	
"padding": 255,	integer	
"compression": "@",	string	
"cryptoMethod": "rsassaPkcs1V15",	string	
"hashAlgorithm": "sha1",	string	
"authenticationDatalength": 4	integer	
}		

Properties

timeout

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

default: 0

key

Specifies the name of a key. The key usage must one of the supported *authenticationAttributes*.

data

The data used to generate the authentication data.

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

ivKey

The name of a key used to decrypt the *iv* or the name of an Initialization Vector (IV).

If *iv* is included, this specifies the name of a key (usage 'K0') used to decrypt the *iv*.

If *iv* is omitted, this specifies the name of an IV (usage 'I0').

This is only used when the *key* usage is a symmetric encryption key and *cryptoMethod* is either CBC or CFB.

iv

The plaintext or encrypted IV for use with the CBC or CFB encryption methods.

If *iv* and *ivKey* properties are omitted the default IV is all zeroes.

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

Properties

padding
Specifies the padding character to use for symmetric key encryption.
Property value constraints:
`minimum: 0`
`maximum: 255`

compression Specifies whether the data is to be compressed (blanks removed) before building the MAC. If this property is omitted compression is not applied. Otherwise this property value is the blank character (e.g. ' ' in ASCII or '@' in EBCDIC). Property value constraints: `pattern: ^[ @]$`
cryptoMethod Specifies the. cryptographic method to use. If the *key* usage is an asymmetric key pair signature usage (e.g. 'S0') this can be one of the following values: • `rsassaPkcs1V15` - Use the RSASSA-PKCS1-v1.5 algorithm. • `rsassaPss` - Use the RSASSA-PSS algorithm. If the *key* usage is a MAC usage (e.g. 'M0') this property should be omitted.
hashAlgorithm Specifies the hash algorithm to use. If the *key* usage is an asymmetric key pair signature usage (e.g. 'S0') this can be one of the following values: • `sha1` - The SHA1 digest algorithm. • `sha256` - The SHA 256 digest algorithm, as defined in ISO/IEC 10118-3:2004 [Ref. crypto-1] and FIPS 180-2 [Ref. crypto-2]. If the *key* usage is a MAC usage (e.g. 'M0') this property should be omitted.

authenticationDatalength
The required authentication data length.
If the *key* usage is an asymmetric key pair signature usage (e.g. 'S0') this property should be omitted.
Property value constraints:
`minimum: 4`
`maximum: 8`

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	string	
`"errorCode": "accessDenied",`	string	
`"authenticationData": "VGhlIG1hYyB2YWx1ZSBv ..."`	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.

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. • `keyNotFound` - The *key* name does not exist. • `keyNoValue` - The *key* name exists but the key is not loaded. • `useViolation` - The *key* usage is not supported. • `modeOfUseNotSupported` - The *key* Mode of Use is not supported. • `invalidKeyLength` - The length of *iv* is not supported or the length of an encryption key is not compatible with the encryption operation required. • `algorithmNotSupported` - The hash algorithm ins not supported. • `cryptoMethodNotSupported` - The cryptographic method specified by *cryptoMethod* is not supported. • `noChipTransactionActive` - A chipcard key is used as encryption key and there is no chip transaction active. active.

authenticationData
The generated authentication data.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

Event Messages

- KeyManagement.DUKPTKSNEvent

10.2.4 Crypto.VerifyAuthentication

This command is used for Message Authentication Code (MAC) and signature verification.

The authentication data is verified using the specified verification attributes. The supported verification attributes are defined in verifyAttributes.

If padding is required, the service will add it using the *padding* parameter. Clients can use an alternative padding method by pre-formatting the data and combining this with the standard padding method.

For MAC verification using the CBC or CFB *cryptoMethod*, the Initialization Vector (*iv*) can be provided as input to this command, or a pre-imported IV referenced by name can be used. The *ivKey* and *iv* are both optional properties.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"key": "Key001",	string	Yes
"data": "R2VuZXJhdGUgYSBNQUMg ...",	string	Yes
"verifyData": "RGF0YSB0byBiZSB2ZXJp ...",	string	Yes
"ivKey": "IVKey",	string	
"iv": "VGhlIGluaXRpYWxpemF0 ...",	string	
"padding": 255,	integer	
"compression": "@",	string	
"cryptoMethod": "rsassaPkcs1V15",	string	
"hashAlgorithm": "sha1"	string	
}		

Properties

timeout

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

default: 0

key

Specifies the name of the verification key. The key usage must one of the supported *verifyAttributes*.

data

The data to be authenticated. The service will generate authentication data (MAC or signature) using the *key*, *cryptoMethod* and "hashAlgorithm* then compare with *verifyData*.

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

verifyData

The authentication data to verify.

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

ivKey

The name of a key used to decrypt the *iv* or the name of an Initialization Vector (IV).

If *iv* is included, this specifies the name of a key (usage 'K0') used to decrypt the *iv*.

If *iv* is omitted, this specifies the name of an IV (usage 'I0').

This is only used when the *key* usage is a symmetric encryption key and *cryptoMethod* is either CBC or CFB.

Properties

iv
The plaintext or encrypted IV for use with the CBC or CFB encryption methods.
If *iv* and *ivKey* properties are omitted the default IV is all zeroes.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

padding
Specifies the padding character to use for symmetric key encryption.
Property value constraints:
`minimum: 0`
`maximum: 255`

compression Specifies whether the data is to be compressed (blanks removed) before building the MAC. If this property is omitted compression is not applied. Otherwise this property value is the blank character (e.g. ' ' in ASCII or '@' in EBCDIC). Property value constraints: `pattern: ^[ @]$`
cryptoMethod Specifies the. cryptographic method to use. If the *key* usage is an asymmetric key pair signature usage (e.g. 'S0') this can be one of the following values: • `rsassaPkcs1V15` - Use the RSASSA-PKCS1-v1.5 algorithm. • `rsassaPss` - Use the RSASSA-PSS algorithm. If the *key* usage is a MAC usage (e.g. 'M0') this property should be omitted.
hashAlgorithm Specifies the hash algorithm to use. If the *key* usage is an asymmetric key pair signature usage (e.g. 'S0') this can be one of the following values: • `sha1` - The SHA1 digest algorithm. • `sha256` - The SHA 256 digest algorithm, as defined in ISO/IEC 10118-3:2004 [Ref. crypto-1] and FIPS 180-2 [Ref. crypto-2]. If the *key* usage is a MAC usage (e.g. 'M0') this property should be omitted.)].

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	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.

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. • `keyNotFound` - The *key* name does not exist. • `keyNoValue` - The *key* name exists but the key is not loaded. • `useViolation` - The *key* usage is not supported. • `modeOfUseNotSupported` - The *key* Mode of Use is not supported. • `invalidKeyLength` - The length of *iv* is not supported or the length of an encryption key is not compatible with the encryption operation required. • `algorithmNotSupported` - The hash algorithm ins not supported. • `cryptoMethodNotSupported` - The cryptographic method specified by *cryptoMethod* is not supported. • `noChipTransactionActive` - A chipcard key is used as encryption key and there is no chip transaction active. active. • `macInvalid` - The MAC verification failed. • `signatureInvalid` - The signature verification failed.

Event Messages

- KeyManagement.DUKPTKSNEvent

10.2.5 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"hashAlgorithm": "sha1",	string	Yes
"data": "U2FtcGxlIERhdGE="	string	
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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.

data

The data to be hashed.

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "accessDenied",	string	
"digest": "OTNjYzE2Y2FkNzYwMTY3 ..."	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.

Properties

digest
Contains the generated digest.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

Event Messages

None

11. Keyboard Interface

This chapter defines the Keyboard interface functionality and messages.

This section describes the general interface for the following functions:

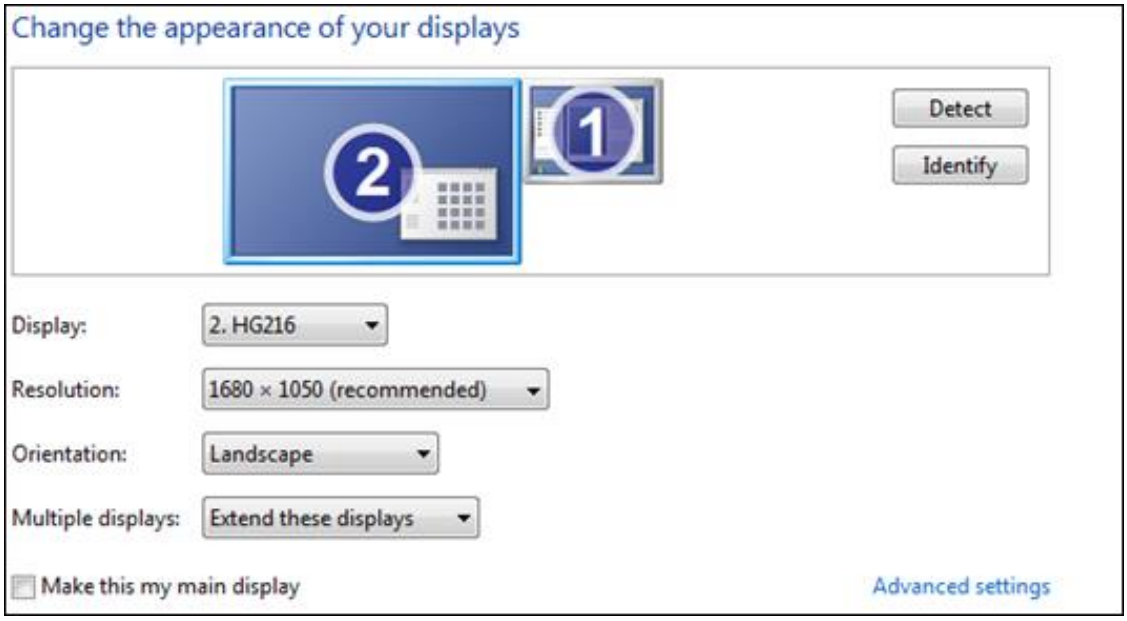
- Entering Personal Identification Numbers (PINs)
- Clear text data handling
- Function key handling
  If the device has local display capability, display handling should be handled using the TextTerminal interface. The adoption of this specification does not imply the adoption of a specific security standard.
- Only numeric PIN pads are handled in this specification.

11.1 General Information

11.1.1 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 operating system desktop, and can be the 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:

- Those which activate touch areas defined by the application.
- Those which activate a random variation of touch areas defined by the application.

The Service Provider, when reporting its capabilities, reports the absolute position of the ETS in 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. This is also the mode that, if supported, is active when none of the other modes are active.

- **Data mode** - a "touch" maps to a 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 a 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 a 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 following commands to enable the touch keyboard definition on the ETS device:

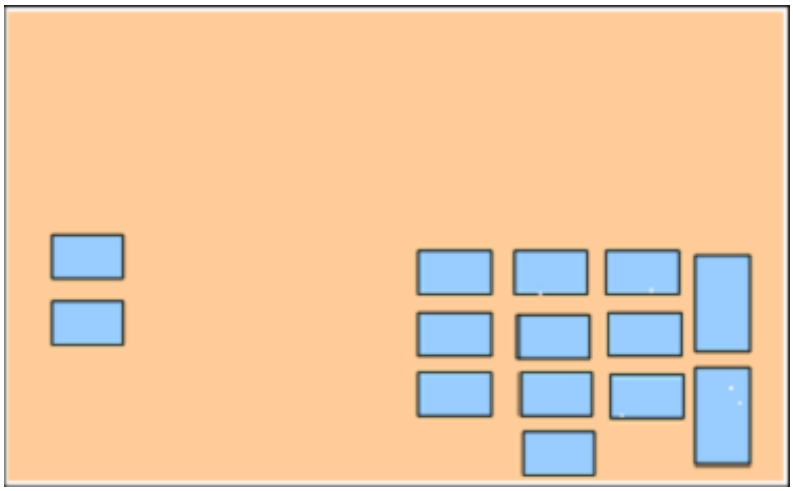
- Keyboard.PinEntry
- Keyboard.DataEntry
- Keyboard.SecureKeyEntry

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 a key value and that key value is not subsequently specified as active in a Keyboard.PinEntry, Keyboard.DataEntry or Keyboard.SecureKeyEntry command, then the Touch Key is made inactive.

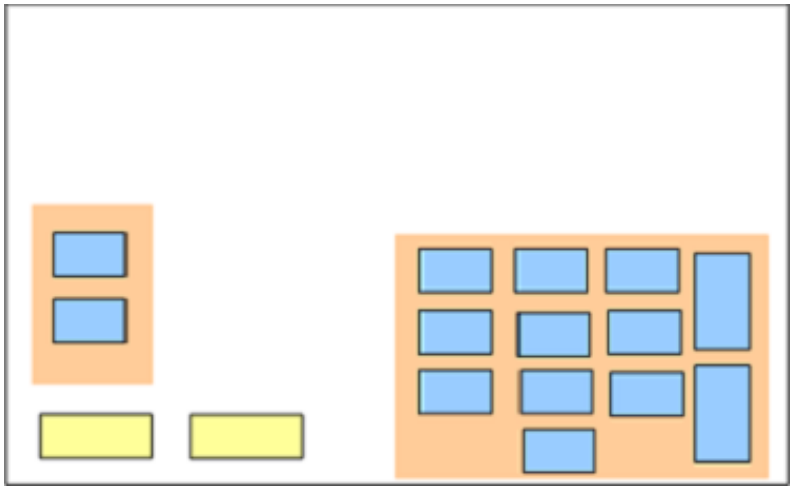
Layouts defined with the Keyboard.DefineLayout command are persistent.

Example 1 – this screen only uses Data mode – the entire screen is a Touch Frame. Mouse mode is not used.

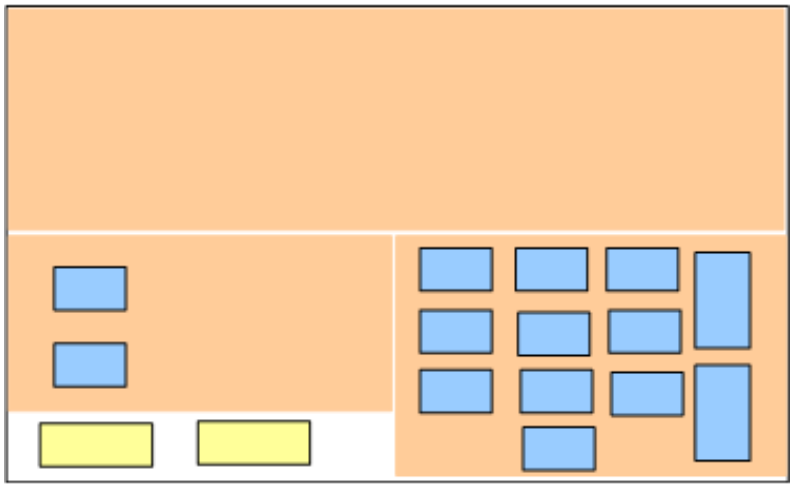


Example 2 – this shows a monitor with two Touch Frames and 14 Touch Keys. The space within the Touch Frames not defined by a Touch Key are inactive (do not respond to touch). All areas outside a Touch Frame operate in Mouse mode. This example shows two Mouse mode "keys". e.g. "Button", HTML "BUTTON" 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.



11.1.2 Layout

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. An ETS can only contain Touch Keys. To determine the frame type, frame xSize and frame ySize should be checked.

The following tables define the possible size and position values that apply to each frame type.

Frame size and position:

Frame Type	**Frame xSize**	**Frame ySize**	**Frame xPos**	**Frame yPos**
Physical Keys on EPP	0	0	0	0
Touch Keys on ETS	> 0	> 0	>= 0	>= 0
Physical Keys on Left Boundary of Screen	0	> 0	0	0
Physical Keys on Right Boundary of Screen	0	> 0	> 0	0
Physical Keys on Top Boundary of Screen	> 0	0	0	0
Physical Keys on Bottom Boundary of Screen	> 0	0	0	> 0

Key size and position:

Frame Type	Key xSize	Key ySize	Key xPos	Key yPos
Physical Keys on EPP	1 to 1000[1]	1 to 1000[2]	0 to 999[3]	0 to 999[4]
Touch Keys on ETS	0 to (Frame *xSize* - Key *xPos*)	0 to (Frame *ySize* - Key *yPos*)	0 to Frame *xSize*	0 to Frame *ySize*
Physical Keys on Left Boundary of Screen	0	0 to (Frame *ySize* - Key *yPos*)	0	0 to Frame *ySize*
Physical Keys on Right Boundary of Screen	0	0 to (Frame *ySize* - Key *yPos*)	Frame *xSize*	0 to Frame *ySize*
Physical Keys on Top Boundary of Screen	0 to (Frame *xSize* - Key *xPos*)	0	0 to Frame *xSize*	0
Physical Keys on Bottom Boundary of Screen	0 to (Frame *xSize* - Key *xPos*)	0	0 to Frame *xSize*	Frame *ySize*

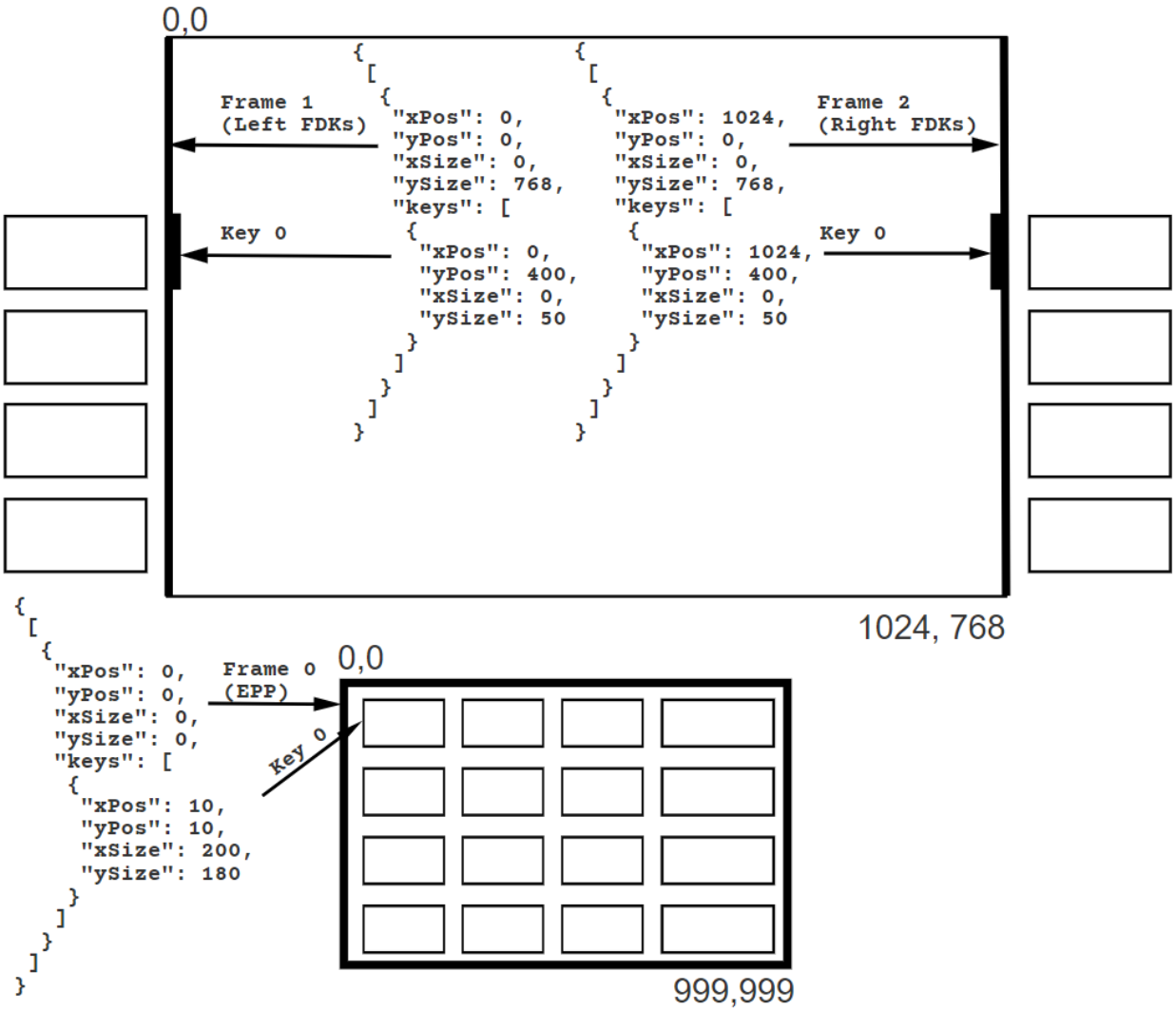
[1]: 1 is the smallest possible size and 1000 is the full width of the frame

[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.



11.2 Command Messages

11.2.1 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:

- Data Entry mode which corresponds to the Keyboard.DataEntry command.
- PIN Entry mode which corresponds to the Keyboard.PinEntry command.
- 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"entryMode": "data"	string	Yes
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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 Keyboard.DataEntry command.
- `pin` - Get the layout for the Keyboard.PinEntry command.
- `secure` - Get the layout for the Keyboard.SecureKeyEntry command.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "modeNotSupported",	string	
"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	Yes

Payload (version 1.0)	Type	Required
"y": false	boolean	Yes
},		
"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)	
See data properties.		
}],		
"secure": [{	array (object)	
See data 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:

- `modeNotSupported` - The specified entry mode is not supported.

layout

Return supported layouts specified by the *entryMode* property.

layout/data

The layout for the Keyboard.DataEntry command.

There can be one or more frames included.

Refer to the layout section for the different types of frames, and see the diagram for an example.

layout/data/xPos

If the frame contains Touch Keys, specifies the left edge of the frame as an offset from the left edge of the screen in pixels and will be less than the width of the screen.

If the frame contains Physical Keys on the boundary of the screen, specifies the left coordinate of the frame as an offset from the left edge of the screen in pixels and will be 0 or the width of the screen in pixels.

If the frame contains Physical Keys not positioned on the screen boundary, this value is 0.

Property value constraints:

```
minimum: 0
```

Properties
layout/data/yPos If the frame contains Touch Keys, specifies the top edge of the frame as an offset from the top edge of the screen in pixels and will be less than the height of the screen. If the frame contains Physical Keys on the boundary of the screen, specifies the top edge of the frame as an offset from the top edge of the screen in pixels and will be 0 or the height of the screen in pixels. If the frame contains Physical Keys not positioned on the screen boundary, this value is 0. Property value constraints: `minimum: 0`
layout/data/xSize If the frame contains Touch Keys, specifies the width of the frame in pixels and will be greater than 0 and less than the width of the screen minus the frame *xPos*. If the frame contains Physical Keys on the boundary of the screen, specifies the width of the frame in pixels and will be 0 or the width of the screen in pixels. If the frame contains Physical Keys not positioned on the screen boundary, this value is 0. Property value constraints: `minimum: 0`
layout/data/ySize If the frame contains Touch Keys, specifies the height of the frame in pixels and will be greater than 0 and less than the height of the screen minus the frame *yPos*. If the frame contains Physical Keys on the boundary of the screen, specifies the height of the frame in pixels and will be 0 or the height of the screen in pixels. If the frame contains Physical Keys not positioned on the screen boundary, this value is 0. Property value constraints: `minimum: 0`
layout/data/float Specifies if the device can float the touch keyboards. This should be omitted not supported.
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 left edge of the key relative to the left side of the frame.
Property value constraints:
`minimum: 0`
`maximum: 999`

layout/data/keys/yPos
Specifies the position of the top edge of the key relative to the top edge of the frame.
Property value constraints:
`minimum: 0`
`maximum: 999`

layout/data/keys/xSize
Specifies the Function Key (FK) width.
Property value constraints:
`minimum: 1`
`maximum: 1000`

Properties

layout/data/keys/ySize
Specifies the Function Key (FK) height.
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 can be omitted if no keys are supported. The following standard values 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 values are also allowed: • `oem[a-zA-Z0-9]*` - A non-standard value Property value constraints: `pattern: ^(zero|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])|oem[a-zA-Z0-9]*)$`
layout/data/keys/shiftKey Specifies the Function Key associated with the physical key in shifted mode. This property can be omitted if no keys are supported. See *key* for the valid property values. Property value constraints: `pattern: ^(zero|one|two|three|four|five|six|seven|eight|nine|[a-f]|enter|cancel|clear|backspace|help|decPoint|shift|doubleZero|tripleZero|fdk(0[1-9]|[12][0-9]|3[0-2])|oem[a-zA-Z0-9]*)$`
layout/pin The layout for the Keyboard.PinEntry command. There can be one or more frames included. Refer to the layout section for the different types of frames, and see the diagram for an example.
layout/secure The layout for the Keyboard.SecureKeyEntry command. There can be one or more frames included. Refer to the layout section for the different types of frames, and see the diagram for an example.

Event Messages

None

11.2.2 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, a notification 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 event 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"minLen": 0,	integer	Yes
"maxLen": 0,	integer	
"autoEnd": false,	boolean	
"echo": "X",	string	
"activeKeys": {	object	
"*one*": {	object	
"terminate": false	boolean	
},		
"*backspace*": {	object	

Payload (version 1.0)	Type	Required
See one properties.		
}		
}		
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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.

Property value constraints:

```
minimum: 0
```

maxLen

Specifies the maximum number of digits which can be entered for the PIN. A value of zero indicates no maximum PIN length verification.

Property value constraints:

```
minimum: 0
```

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.

default: false

echo

Specifies the replace character to be echoed on a local display for the PIN digit.

Property value constraints:

```
minLength: 1
maxLength: 1
```

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: • `oem[a-zA-Z0-9]*` - A non-standard key name
activeKeys/one (example name) Property name constraints: `pattern: ^(zero|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])|oem[a-zA-Z0-9]*)$`
activeKeys/one/terminate The key is a terminate key. default: false

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	string	
`"errorCode": "keyInvalid",`	string	
`"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.

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: • `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 Property value constraints: `minimum: 0`
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

11.2.3 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 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"maxLen": 0,	integer	
"autoEnd": false,	boolean	
"activeKeys": {	object	
"*one*": {	object	
"terminate": false	boolean	
},		
"*backspace*": {	object	
See one properties.		
}		
}		
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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.

Property value constraints:

```
minimum: 0
```

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.

default: false

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:

- `oem[a-zA-Z0-9]*` - A non-standard key name

activeKeys/one (example name)

Property name constraints:

```
pattern: ^(zero|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])|oem[a-zA-Z0-9]*)$
```

activeKeys/one/terminate

The key is a terminate key.

default: false

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	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
Property value constraints:
`minimum: 0`

pinKeys
Array contains the keys entered by the user

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` - A 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).

Properties

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.

The following standard values 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 values are also allowed:

- `oem[a-zA-Z0-9]*` - A non-standard value

Property value constraints:

```
pattern: ^(zero|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])|oem[a-zA-Z0-9]*)$
```

completion

Specifies the reason for completion of the entry.

Event Messages

- Keyboard.KeyEvent
- Keyboard.EnterDataEvent
- Keyboard.LayoutEvent

11.2.4 Keyboard.Reset

Sends a service reset to the Service Provider.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

Properties

completionCode
The completion code.

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

Event Messages

None

11.2.5 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'), a notification 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 a 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"keyLen": 0,	integer	Yes
"autoEnd": false,	boolean	
"activeKeys": {	object	Yes
"*one*": {	object	
"terminate": false	boolean	
},		
"*backspace*": {	object	
See one properties.		
}		

Payload (version 1.0)	Type	Required
`},`		
`"verificationType": "self",`	string	Yes
`"cryptoMethod": "des"`	string	
`}`		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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 single-length 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

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:

- `oem[a-zA-Z0-9]*` - A non-standard key name

Properties
activeKeys/one (example name) Property name constraints: `pattern: ^(zero|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])|oem[a-zA-Z0-9]*)$`
activeKeys/one/terminate The key is a terminate key. default: false
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 (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	string	
`"errorCode": "accessDenied",`	string	
`"digits": 0,`	integer	
`"completion": "auto",`	string	
`"kcv": "S2V5IENoZWNrIFZhbHVl"`	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.

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. Property value constraints: `minimum: 0`
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` - A 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 Base64. 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.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

Event Messages

- Keyboard.KeyEvent
- Keyboard.EnterDataEvent
- Keyboard.LayoutEvent

11.2.6 Keyboard.KeypressBeep

This command is used to enable or disable the device from emitting a beep tone on subsequent key presses of active or inactive 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 (version 1.0)	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 0, 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 inactive key subsequently pressed on the PIN. This selectively turns beeping on and off for active, inactive or both types of keys.

mode/active
Specifies whether beeping should be enabled for active keys.
default: false

mode/inactive
Specifies whether beeping should be enabled for inactive keys.
default: false

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

Properties

completionCode
The completion code.

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

Event Messages

None

11.2.7 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:

- Mouse mode.
- Data mode which corresponds to the Keyboard.DataEntry command.
- PIN mode which corresponds to the Keyboard.PinEntry command.
- 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.

It is recommended that the Keyboard.GetLayout command is used before this command to check for the presence of frames containing Physical Keys (FKs or FDKs). If a layout includes one or more frames containing Physical Keys, the number of frames containing Physical Keys, the size and position of the frame, and the size, position and order of the keys contained in the frame, cannot be changed.

Layouts defined with this command are persistent.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"layout": {	object	Yes
"data": [{	array (object)	
"xPos": 0,	integer	Yes
"yPos": 0,	integer	Yes
"xSize": 0,	integer	Yes
"ySize": 0,	integer	Yes
"float": {	object	
"x": false,	boolean	Yes
"y": false	boolean	Yes
},		
"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)	

Payload (version 1.0)	Type	Required
See data properties.		
}],		
"secure": [{	array (object)	
See data properties.		
}]		
}		
}		

Properties

timeout

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

default: 0

layout

Specify layouts to define.

layout/data

The layout for the Keyboard.DataEntry command.

There can be one or more frames included.

Refer to the layout section for the different types of frames, and see the diagram for an example.

layout/data/xPos

If the frame contains Touch Keys, specifies the left edge of the frame as an offset from the left edge of the screen in pixels and will be less than the width of the screen.

If the frame contains Physical Keys on the boundary of the screen, specifies the left coordinate of the frame as an offset from the left edge of the screen in pixels and will be 0 or the width of the screen in pixels.

If the frame contains Physical Keys not positioned on the screen boundary, this value is 0.

Property value constraints:

```
minimum: 0
```

layout/data/yPos

If the frame contains Touch Keys, specifies the top edge of the frame as an offset from the top edge of the screen in pixels and will be less than the height of the screen.

If the frame contains Physical Keys on the boundary of the screen, specifies the top edge of the frame as an offset from the top edge of the screen in pixels and will be 0 or the height of the screen in pixels.

If the frame contains Physical Keys not positioned on the screen boundary, this value is 0.

Property value constraints:

```
minimum: 0
```

layout/data/xSize

If the frame contains Touch Keys, specifies the width of the frame in pixels and will be greater than 0 and less than the width of the screen minus the frame *xPos*.

If the frame contains Physical Keys on the boundary of the screen, specifies the width of the frame in pixels and will be 0 or the width of the screen in pixels.

If the frame contains Physical Keys not positioned on the screen boundary, this value is 0.

Property value constraints:

```
minimum: 0
```

Properties
layout/data/ySize If the frame contains Touch Keys, specifies the height of the frame in pixels and will be greater than 0 and less than the height of the screen minus the frame *yPos*. If the frame contains Physical Keys on the boundary of the screen, specifies the height of the frame in pixels and will be 0 or the height of the screen in pixels. If the frame contains Physical Keys not positioned on the screen boundary, this value is 0. Property value constraints: `minimum: 0`
layout/data/float Specifies if the device can float the touch keyboards. This should be omitted not supported.
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 left edge of the key relative to the left side of the frame.
Property value constraints:
`minimum: 0`
`maximum: 999`

layout/data/keys/yPos
Specifies the position of the top edge of the key relative to the top edge of the frame.
Property value constraints:
`minimum: 0`
`maximum: 999`

layout/data/keys/xSize
Specifies the Function Key (FK) width.
Property value constraints:
`minimum: 1`
`maximum: 1000`

layout/data/keys/ySize
Specifies the Function Key (FK) height.
Property value constraints:
`minimum: 1`
`maximum: 1000`

Properties
layout/data/keys/key Specifies the Function Key associated with the physical area in non-shifted mode. This property can be omitted if no keys are supported. The following standard values 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 values are also allowed: • `oem[a-zA-Z0-9]*` - A non-standard value Property value constraints: `pattern: ^(zero|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])|oem[a-zA-Z0-9]*)$`
layout/data/keys/shiftKey Specifies the Function Key associated with the physical key in shifted mode. This property can be omitted if no keys are supported. See *key* for the valid property values. Property value constraints: `pattern: ^(zero|one|two|three|four|five|six|seven|eight|nine|[a-f]|enter|cancel|clear|backspace|help|decPoint|shift|doubleZero|tripleZero|fdk(0[1-9]|[12][0-9]|3[0-2])|oem[a-zA-Z0-9]*)$`
layout/pin The layout for the Keyboard.PinEntry command. There can be one or more frames included. Refer to the layout section for the different types of frames, and see the diagram for an example.
layout/secure The layout for the Keyboard.SecureKeyEntry command. There can be one or more frames included. Refer to the layout section for the different types of frames, and see the diagram for an example.

Completion Message

Payload (version 1.0)	Type	Required
{		

Payload (version 1.0)	Type	Required
"completionCode": "success",	string	
"errorDescription": "Device not available",	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.

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 values for *key* and *shiftKey* can only be used once per layout.

Event Messages

None

11.3 Event Messages

11.3.1 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.

Event Message

Payload (version 1.0)	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` - A 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).

Properties

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.

The following standard values 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 values are also allowed:

- `oem[a-zA-Z0-9]*` - A non-standard value

Property value constraints:

```
pattern: ^(zero|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])|oem[a-zA-Z0-9]*)$
```

11.3.2 Keyboard.EnterDataEvent

This mandatory event notifies the application when the device is ready for the user to start entering data.

Event Message

Payload (version 1.0)	Type	Required
{		
}		

11.3.3 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).

Event Message

Payload (version 1.0)	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	Yes
"y": false	boolean	Yes
},		
"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)	
See data properties.		
}],		
"secure": [{	array (object)	
See data properties.		
}]		
}		

Properties

data

The layout for the Keyboard.DataEntry command.

There can be one or more frames included.

Refer to the layout section for the different types of frames, and see the diagram for an example.

Properties
data/xPos If the frame contains Touch Keys, specifies the left edge of the frame as an offset from the left edge of the screen in pixels and will be less than the width of the screen. If the frame contains Physical Keys on the boundary of the screen, specifies the left coordinate of the frame as an offset from the left edge of the screen in pixels and will be 0 or the width of the screen in pixels. If the frame contains Physical Keys not positioned on the screen boundary, this value is 0. Property value constraints: `minimum: 0`
data/yPos If the frame contains Touch Keys, specifies the top edge of the frame as an offset from the top edge of the screen in pixels and will be less than the height of the screen. If the frame contains Physical Keys on the boundary of the screen, specifies the top edge of the frame as an offset from the top edge of the screen in pixels and will be 0 or the height of the screen in pixels. If the frame contains Physical Keys not positioned on the screen boundary, this value is 0. Property value constraints: `minimum: 0`
data/xSize If the frame contains Touch Keys, specifies the width of the frame in pixels and will be greater than 0 and less than the width of the screen minus the frame *xPos*. If the frame contains Physical Keys on the boundary of the screen, specifies the width of the frame in pixels and will be 0 or the width of the screen in pixels. If the frame contains Physical Keys not positioned on the screen boundary, this value is 0. Property value constraints: `minimum: 0`
data/ySize If the frame contains Touch Keys, specifies the height of the frame in pixels and will be greater than 0 and less than the height of the screen minus the frame *yPos*. If the frame contains Physical Keys on the boundary of the screen, specifies the height of the frame in pixels and will be 0 or the height of the screen in pixels. If the frame contains Physical Keys not positioned on the screen boundary, this value is 0. Property value constraints: `minimum: 0`
data/float Specifies if the device can float the touch keyboards. This should be omitted not supported.
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 left edge of the key relative to the left side of the frame.
Property value constraints:
`minimum: 0`
`maximum: 999`

Properties

data/keys/yPos

Specifies the position of the top edge of the key relative to the top edge of the frame.

Property value constraints:

```
minimum: 0
maximum: 999
```

data/keys/xSize

Specifies the Function Key (FK) width.

Property value constraints:

```
minimum: 1
maximum: 1000
```

data/keys/ySize

Specifies the Function Key (FK) height.

Property value constraints:

```
minimum: 1
maximum: 1000
```

data/keys/key

Specifies the Function Key associated with the physical area in non-shifted mode. This property can be omitted if no keys are supported.

The following standard values 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 values are also allowed:

- `oem[a-zA-Z0-9]*` - A non-standard value

Property value constraints:

```
pattern: ^(zero|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])|oem[a-zA-Z0-9]*)$
```

Properties
data/keys/shiftKey Specifies the Function Key associated with the physical key in shifted mode. This property can be omitted if no keys are supported. See *key* for the valid property values. Property value constraints: `pattern: ^(zero|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])|oem[a-zA-Z0-9]*)$`
pin The layout for the Keyboard.PinEntry command. There can be one or more frames included. Refer to the layout section for the different types of frames, and see the diagram for an example.
secure The layout for the Keyboard.SecureKeyEntry command. There can be one or more frames included. Refer to the layout section for the different types of frames, and see the diagram for an example.

12. PinPad Interface

This chapter defines the PinPad interface functionality and messages.

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

12.1 General Information

12.1.1 References

ID	Description
pinpad-1	SHA-1 Hash algorithm ANSI X9.30-2:1993, Public Key Cryptography for Financial Services Industry Part2
pinpad-2	ANSI X3.92, American National Standard for Data Encryption Algorithm (DEA), American National Standards Institute, 1983
pinpad-3	ANSI X9.8-1995, Banking – Personal Identification Number Management and Security, Part 1 + 2, American National Standards Institute
pinpad-4	IBM, Common Cryptographic Architecture: Cryptographic Application Programming Interface, SC40-1675-1, IBM Corp., Nov 1990
pinpad-5	Oliself2 Specifiche Tecniche, PIN Block Detail for FormAp
pinpad-6	ANSI X9.24-1:2009, Retail Financial Services Symmetric Key Management Part 1: Using Symmetric Techniques
pinpad-7	NIST Special Publication 800-38A: Recommendation for Block Cipher Modes of Operation
pinpad-8	NIST Special Publication 800-38E: Recommendation for Block Cipher Modes of Operation: the XTS-AES Mode for Confidentiality on Storage Devices

12.2 Command Messages

12.2.1 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 does not imply in anyway the certification level achieved by the device.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"manufacturerIdentifier": "Manufacturer ID",	string	
"modelIdentifier": "Model ID",	string	
"hardwareIdentifier": "Hardware ID",	string	
"firmwareIdentifier": "Formware ID",	string	
"applicationIdentifier": "Application ID"	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.

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.

Properties
applicationIdentifier Returns the application identifier of the PIN device. This value is not set if the application identifier is not available.

Event Messages

None

12.2.2 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.MaintainPin command.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"validationData": "0812746533758375",	string	Yes
"offset": "0000000000000000",	string	
"padding": 00,	undefined	Yes
"maxPIN": 0,	integer	Yes
"valDigits": 0,	integer	Yes
"noLeadingZero": false,	boolean	Yes
"key": "Key01",	string	Yes
"keyEncKey": "Key02",	string	
"decTable": "3183042102277795"	string	Yes
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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

maxPIN

Maximum number of PIN digits to be used for validation. This property corresponds to PINMINL in the IBM 3624 specification (see [Ref. pinpad-4]).

Property value constraints:

```
minimum: 0
```

Properties
valDigits Number of Validation digits from the validation data to be used for validation. This is the length of the *validationData*. Property value constraints: `minimum: 0`
noLeadingZero If 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 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 (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	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.
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 Specifies whether the PIN is correct or not.

Event Messages

- KeyManagement.IllegalKeyAccessEvent

12.2.3 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 using the PinPad.MaintainPin command.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"pan": "01234567890123456789123",	string	Yes
"pvv": "0286",	string	Yes
"key": "Key01",	string	Yes
"keyEncKey": "Key02"	string	
}		

Properties

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

pan
Primary Account Number from track data, as an ASCII string. The 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 (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	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.
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 Specifies whether the PIN is correct or not.

Event Messages

- KeyManagement.IllegalKeyAccessEvent

12.2.4 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 using the PinPad.MaintainPin command.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"presentAlgorithm": "presentClear",	string	Yes
"chipProtocol": "chipT0",	string	Yes
"chipData": "Y2hpcCBkYXRhIHRvIHNl ...",	string	Yes
"algorithmData": {	object	Yes
"pinPointer": 0,	integer	Yes
"pinOffset": 0	integer	Yes
}		
}		

Properties

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

presentAlgorithm
Specifies the algorithm that is used for presentation. See presentationAlgorithms for possible values.

chipProtocol
Identifies the protocol that is used to communicate with the chip. See chipProtocols for possible values.

chipData
The data to be sent to the chip.
Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

algorithmData
Contains the data required for the specified presentation algorithm.

algorithmData/pinPointer
The byte offset where to start inserting the PIN into chipData. The leftmost byte is numbered zero.
Property value constraints:

```
minimum: 0
```

algorithmData/pinOffset
The bit offset within the byte specified by *pinPointer* property where to start inserting the PIN. The leftmost bit numbered zero.
Property value constraints:

```
minimum: 0
```

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "accessDenied",	string	
"chipProtocol": "chipT0",	string	
"chipData": "Y2hpcCBkYXRhIHJlY2Vp ..."	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.
- `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 data returned from the chip.

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

Event Messages

None

12.2.5 PinPad.Reset

Sends a service reset to the Service.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

Properties

completionCode
The completion code.

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

Event Messages

None

12.2.6 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"maintainPIN": false	boolean	
}		

Properties

timeout
Timeout in milliseconds for the command to complete. If set to 0, 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 (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

Properties

completionCode
The completion code.

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

Event Messages

None

12.2.7 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

If this command is required and it is not called, the *Keyboard.GetPin* command will fail with the generic error sequenceError.

If the input parameters passed to this command and PinPad.GetPinBlock are not identical, the PinPad.GetPinBlock command will fail with the generic error invalidData.

The data associated with this command will be cleared on a PinPad.GetPinBlock command.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"customerData": "9385527846382726",	string	Yes
"xorData": "0123456789ABCDEF",	string	Yes
"padding": 2,	integer	Yes
"format": "ibm3624",	string	Yes
"key": "PinKey01",	string	
"secondEncKey": "Key01",	string	
"cryptoMethod": "ecb"	string	
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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 (See [Ref. pinpad-1], [Ref. pinpad-2], [Ref. pinpad-3]) 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. If this value is omitted no XOR-operation will be performed.

The format is a string of case-insensitive hexadecimal data.

If the formatted PIN is not encrypted twice (i.e. if the secondEncKey property is omitted) this is ignored.

Property value constraints:

```
pattern: ^[0-9a-fA-F]{2,}$
```

Properties

padding
Specifies the padding character. This property is not applicable for PIN block formats with fixed, sequential or random padding and can be omitted.
Property value constraints:
`minimum: 0`
`maximum: 15`

format Specifies the format of the PIN block. For a list of valid values see pinFormats.
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.
cryptoMethod This specifies the cryptographic method to be used for this command, this property is not required if no encryption is required. For a list of valid values see cryptoMethod. If specified, this must be compatible with the key identified by *key*.

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available"`	string	
`}`		

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

Event Messages

- KeyManagement.IllegalKeyAccessEvent

12.2.8 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.MaintainPin command.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"customerData": "9385527846382726",	string	Yes
"xorData": "0123456789ABCDEF",	string	Yes
"padding": 2,	integer	Yes
"format": "ibm3624",	string	Yes
"key": "PinKey01",	string	
"secondEncKey": "Key01",	string	
"cryptoMethod": "ecb"	string	
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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 (See [Ref. pinpad-1], [Ref. pinpad-2], [Ref. pinpad-3]) 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. If this value is omitted no XOR-operation will be performed.

The format is a string of case-insensitive hexadecimal data.

If the formatted PIN is not encrypted twice (i.e. if the secondEncKey property is omitted) this is ignored.

Property value constraints:

```
pattern: ^[0-9a-fA-F]{2,}$
```

Properties

padding
Specifies the padding character. This property is not applicable for PIN block formats with fixed, sequential or random padding and can be omitted.
Property value constraints:
`minimum: 0`
`maximum: 15`

format Specifies the format of the PIN block. For a list of valid values see pinFormats.
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.
cryptoMethod This specifies the cryptographic method to be used for this command, this property is not required if no encryption is required. For a list of valid values see cryptoMethod. If specified, this must be compatible with the key identified by *key*.

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	string	
`"errorCode": "keyNotFound",`	string	
`"pinBlock": "UGluYmxvY2sgZGF0YQ=="`	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.

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. • `cryptoMethodNotSupported` - The cryptographic method specified by cryptoMethod is not supported.

pinBlock
The encrypted PIN block.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

Event Messages

- KeyManagement.IllegalKeyAccessEvent
- KeyManagement.DUKPTKSNEvent

13. Printer Interface

This chapter defines the Printer interface functionality and messages.

This specification describes the functionality of the services provided by banking printers and scanning devices under XFS4IoT, focusing on the following areas:

- application programming for printing
- print document definition
- scanning images for devices such as check scanners

The XFS4IoT Printer interface is implemented around a forms model which also standardizes the basic document definition.

13.1 General Information

13.1.1 References

ID	Description
printer-1	ANSI X3.17-1981: Character Set For Optical Character Recognition (OCR-A)
printer-2	ANSI X3.49-1975: Character Set For Optical Character Recognition (OCR-B)

13.1.2 Banking Printer Types

The XFS4IoT 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 XFS4IoT 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 class interface can also be used for devices that are capable of scanning without necessarily printing. This includes devices such as Check Scanners.

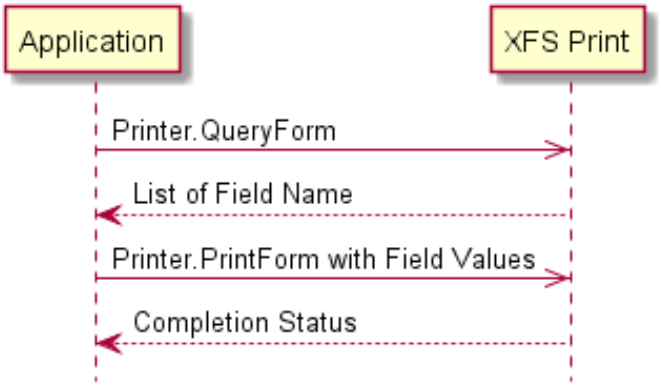
The specification refers to the terms paper and media. When the term paper is used this refers to paper that is situated in a paper supply attached to the device. The term media is used for media that is inserted by the customer (e.g. check and other material that is scanned) or that is issued to the customer (e.g. a receipt or statement). Receipt, document and passbook printers with white passbook dispensing capability have both. As soon as the paper gets printed it becomes media. Scanners only have media. The term media does not apply to journal printers. When paper is in the print position it is classified as media, on some printers that maintain paper under the print head there will always be both media and paper.

13.1.3 Forms Model

The XFS4IoT printing service 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 XFS4IoT 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 XFS4IoT 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 XFS4IoT 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.PrintRaw command.

The document definition under XFS4IoT 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 XFS4IoT 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.

13.1.4 Command Overview

The basic operation of the print devices is managed using the two primary commands:

- **Printer.GetQueryForm** This command retrieves the form header information, and the list of fields.
- **Printer.PrintForm** This command includes as parameter data the name of the form to select and the required field data values.

This approach combines in the most efficient manner the four logical steps required to print a form:

- Selecting a document form object.
- Querying the service for the list of fields.
- Supplying the data for each field.
- Issuing the print command.

By using *Printer.GetQueryForm* 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.PrintNative command can be used to print data that contains a complete print job in the native printer language. This data will have been created using the native Operating System API (for example, Windows GDI).

13.1.5 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 and font names are case sensitive.
- **Strings**
  All strings must be enclosed in double quote characters ("). Standard C escape sequences are allowed.
- **Comments**
  Start with two forward slashes (//), end at line termination

Other Notes:

- The values of a keyword are separated by commas.
- If a keyword is present, all its values must be specified; default values are used only if the keyword is absent.
- Values that are character strings are marked with * in the definitions below and must be quoted as specified above.
- The order of attributes within the forms is not mandatory and the attributes may be defined in any order.
- 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 subform name 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 frame name are invalid. The formInvalid error will be returned if specified in the input to the command.
- All definitions must be encoded in UTF-8. Keywords are restricted to an internal representation of ISO 646 (ANSI) characters. Values for the INITIALVALUE and FORMAT keywords can have UNICODE values.

Form and Media Measurements

The UNIT keyword sections of the form and media definitions specify the base horizontal and vertical resolution as follows:

- The *base* value specifies the base unit of measurement.
- The *x* and *y* values specify the horizontal and vertical resolution as fractions of the base value (e.g. an *x* value of 10 and a base value of MM means that the base horizontal resolution is 0.1 mm).

The base resolutions thus defined by the UNIT keyword section of the XFSFORM definition are used as the units of the form definition keyword sections:

- SIZE (*width* and *height* values)
- ALIGNMENT (*xoffset* and *yoffset* values)

and of the sub-form definition keyword sections:

- POSITION (*x* and *y* values)
- SIZE (*width* and *height* values)

and of the field definition keyword sections:

- POSITION (*x* and *y* values)
- SIZE (*width* and *height* values)
- INDEX (*xoffset* and *yoffset* values)

and of the frame definition keyword sections:

- POSITION (*x* and *y* values)
- SIZE (*width* and *height* values)
- REPEATONX (*xoffset* value)
- REPEATONY (*yoffset* value)

The base resolutions thus defined by the UNIT keyword section of the XFSMEDIA definition are used as the units of the media definition keyword sections:

- SIZE (*width* and *height* values)
- PRINTAREA (*x*, *y*, *width* and *height* values)
- RESTRICTED (*x*, *y*, *width* and *height* values)

NOTE: The origin for coordinate based systems is (0,0). The origin for row/column based systems can be (0,0) or (1,1) and must be configurable within the service.

Form Definition

Attributes are not required in any mandatory order within a Form definition.

Keyword	Nested Keyword	Required	Names	Notes
XFSFORM		✓	*formname**	
BEGIN		✓		
	UNIT	✓	*base,x,y*	**base** - Base resolution unit for form definition: **MM**, **INCH** or **ROWCOLUMN** **x** - Horizontal base unit fraction **y** - Vertical base unit fraction
	SIZE	✓	*width,height*	**width** - Width of form **height** - Height of form

Keyword	Nested Keyword	Required	Names	Notes
	ALIGNMENT		*alignment,xoffset,yoffset*	**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 the form: **PORTRAIT** (default) or **LANDSCAPE**
	SKEW		*skewfactor*	Maximum skew factor in degrees (default = 0)
	VERSION		*major,minor,date*,author**	**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.
	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

Keyword	Nested Keyword	Required	Names	Notes
	COMMENT		*comment**	Comment section
	USERPROMPT		*prompt**	Prompt string for user interaction
	[XFSFIELD **BEGIN** ... **END]**		*fieldname**	One field definition 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 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 for each subform in the form. The *subformname* within a form must be unique.
END		✓		

SubForm Definition

Attributes are not required in any mandatory order within a SubForm definition.

Keyword	Nested Keyword	Required	Names	Notes
XFSSUBFORM		✓	*subformname**	
BEGIN		✓		
	POSITION	✓	*X,Y* or *X,Y,Z*	**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.
	SIZE	✓	*width,height*	**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 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 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

Keyword	Nested Keyword	Required	Names	Notes
XFSFIELD		✓	*fieldname**	The fieldname within a form and its optional subforms must be unique.
BEGIN		✓		
	POSITION	✓	*X*,*Y* or *X*,*Y*,*Z*	**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.

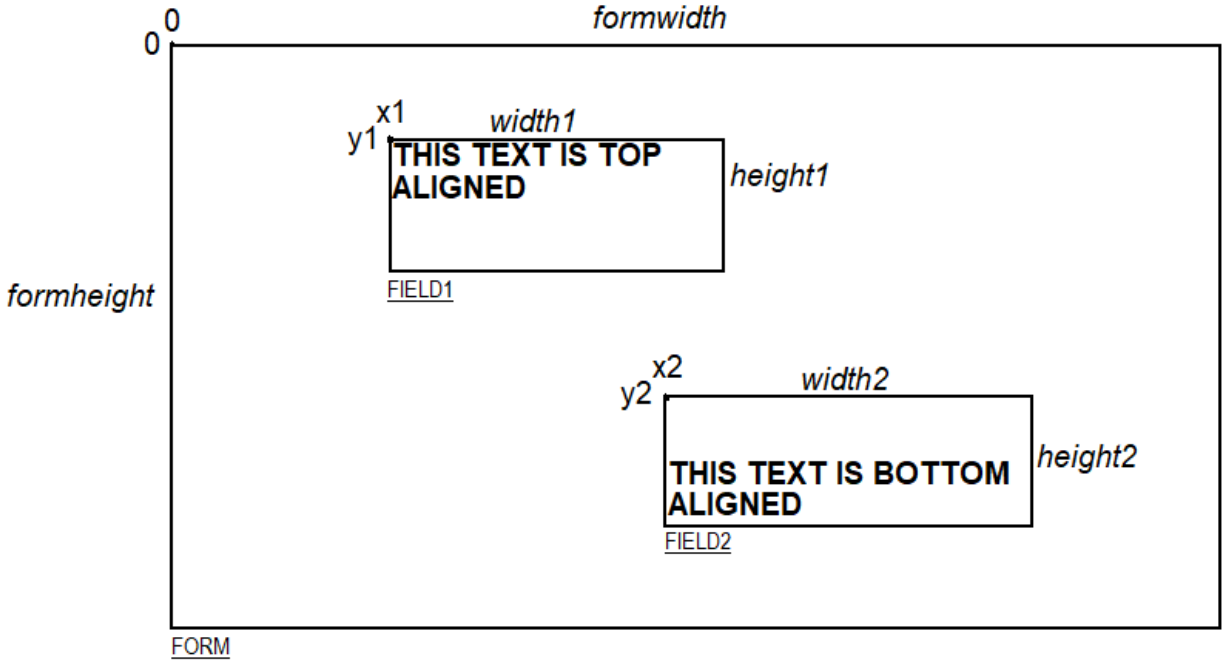
Keyword	Nested Keyword	Required	Names	Notes
	HEADER		*N, N-N* or *ALL*	This field is either a form or subform header field. **N** represents a form/subform page number (relative to 0) the header field is to print within. **N-N** represents a form/subform page number range the header field is to print within. Combinations of N and N-N may exist separated by commas. **ALL** 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, N-N* or *ALL*	This field is either a form or subform footer field. **N** represents a form/subform page number (relative to 0) the footer field is to print within. **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** 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) or **BACK**
	SIZE	✓	*width,height*	**width** - Field width. **height** - Field height.
	INDEX		*repeatcount,xoffset,yoffset*	**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.

Keyword	Nested Keyword	Required	Names	Notes
	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 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 fits the data into the field as well as it can) OVERWRITE (a contiguous write) WORDWRAP

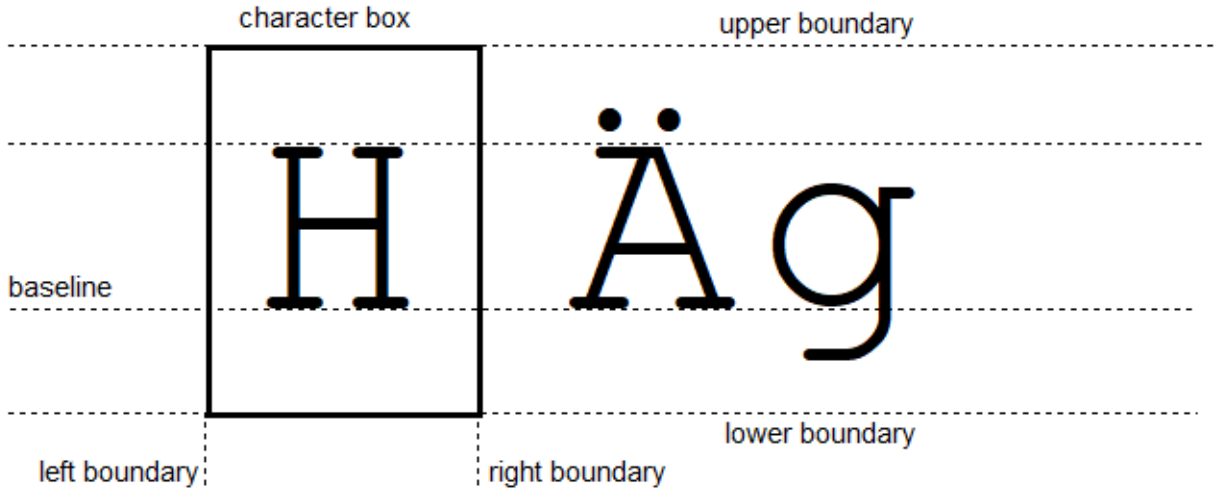
Keyword	Nested Keyword	Required	Names	Notes
	STYLE		*style*	Display attributes as a combination, using the | operator, of the following: NORMAL (default) BOLD ITALIC UNDER (single underline) DOUBLEUNDER (double underline) DOUBLE (double width) TRIPLE (triple width) QUADRUPLE (quadruple width) STRIKETHROUGH 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

Keyword	Nested Keyword	Required	Names	Notes
	COLOR		*color*	Color name: 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.
	FONT		*fontname**	Font name: This attribute is interpreted by the service. 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. For GRAPHIC fields, this defines the type of the graphic, for example, "BMP", "PNG" etc.
	INITIALVALUE		*value**	Initial value. For GRAPHIC type fields, this value may contain Base64 encoded image data. The type of this graphic will be determined by the FORMAT field.
END		✓		

The following diagrams illustrate the positioning and sizing of text fields on a form, and the vertical alignment of text within a field using **VERTICAL=TOP** and **VERTICAL=BOTTOM** values in the field definition.



Definition of the character drawing box:



When more than one line of text is to be printed in a field, and the definition includes **VERTICAL=BOTTOM**, the vertical position of the first line is calculated using the specified (or implied) **LPI** value.

Frame Definition

Keyword	Nested Keyword	Required	Names	Notes
XFSFRAME		✓	*framename**	
BEGIN		✓		

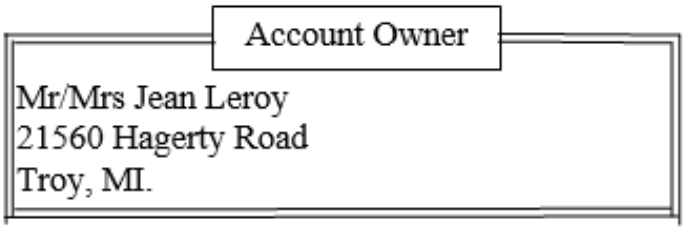
Keyword	Nested Keyword	Required	Names	Notes
	POSITION	✓	*X,Y* or *X,Y,Z*	**X** - Horizontal position of top left corner of the frame (relative to left side of form/subform). **Y** or **Y,Z** - Vertical position of top left corner of frame (relative to top of form/subform). Format **Y,Z** is used to indicate 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*, *N-N* or *ALL*	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** 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** 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.

Keyword	Nested Keyword	Required	Names	Notes
	FOOTER		*N, N-N* or *ALL*	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** 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** 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
	SIZE	✓	*width,height*	**width** - Frame width in base horizontal units for the form. **height** - Frame height in base vertical units for the form.
	REPEATONX		*repeatcount,xoffset*	**repeatcount** - Count how often this frame is repeated horizontally in the form. **xoffset** - Horizontal offset for next frame in base horizontal units.
	REPEATONY		*repeatcount,yoffset*	**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 fits the frame into the media as well as it can)

Keyword	Nested Keyword	Required	Names	Notes
	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
	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

Keyword	Nested Keyword	Required	Names	Notes
	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, Services 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, Services 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.

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, Services 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 or the device it controls is not mandatory to be XFS4IoT compliant.

Sample 1 - Simple Framing

The form:

```
XFSFORM "Multiple Balances"
BEGIN
  UNIT INCH, 16, 16
  SIZE 91, 64
  VERSION 1, 0, "13/09/96", "XFS"
  XFSFIELD "Account Title"
  BEGIN
    POSITION 15, 4
    SIZE 30, 4
    CLASS STATIC
    HORIZONTAL CENTER
    INITIALVALUE "Account"
  END
  XFSFIELD "Balance Title"
  BEGIN
    POSITION 45, 4
    SIZE 30, 4
    CLASS STATIC
    HORIZONTAL CENTER
    INITIALVALUE "Balance"
  END
  XFSFIELD "Account"
  BEGIN
    POSITION 15, 8
    SIZE 30, 4
    INDEX 10, 0, 3
  END //"Account"
  XFSFIELD "Balance"
  BEGIN
    POSITION 45, 8
    SIZE 30, 4
    INDEX 10, 0, 3
    HORIZONTAL RIGHT
  END //"Balance"
  XFSFRAME "Account Title"
  BEGIN
    POSITION 15, 4
    FRAMES "Account Title"
    SIZE 30, 4
    STYLE DOUBLE_THIN
  END
  XFSFRAME "Balance Title"
  BEGIN
    POSITION 45, 4
    FRAMES "Balance Title"
    SIZE 30, 4
    STYLE DOUBLE_THIN
  END
  XFSFRAME "Account"
  BEGIN
    POSITION 15, 8
    FRAMES "Account"
    SIZE 30, 34
    STYLE DOUBLE_THIN
  END
  XFSFRAME "Balance"
  BEGIN
    POSITION 45, 8
    FRAMES "Balance"
    SIZE 30, 34
    STYLE DOUBLE_THIN
  END
END
```

When printed with the following field list:

```
Account[0]=0123456789123001
Account[1]=0123456789123002
Account[2]=0123456789123003
Balance[0]=$17465.12
Balance[1]=$2458.23
Balance[2]=$6542.78
```

Will print:

Account	Balance
0123456789123001	$17465.12
0123456789123002	$2458.23
0123456789123003	$6542.78

Sample 2 - Framing With Title

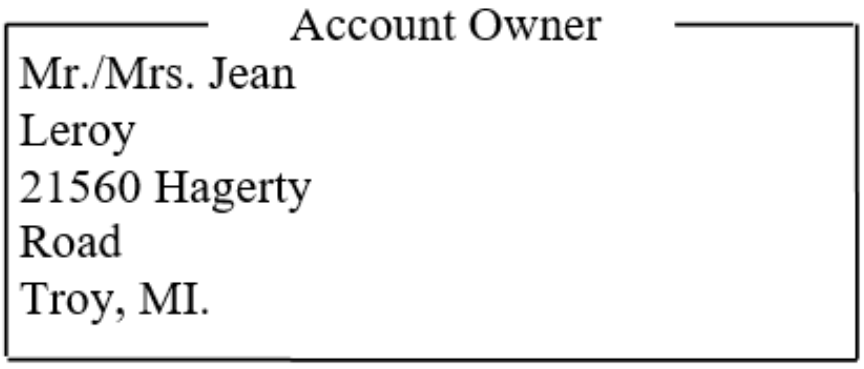
The form:

```
XFSFORM "Bank Details"
BEGIN
  UNIT INCH, 16, 16
  SIZE 121, 64
  VERSION 1, 0, "13/09/96", "XFS Editor"
  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

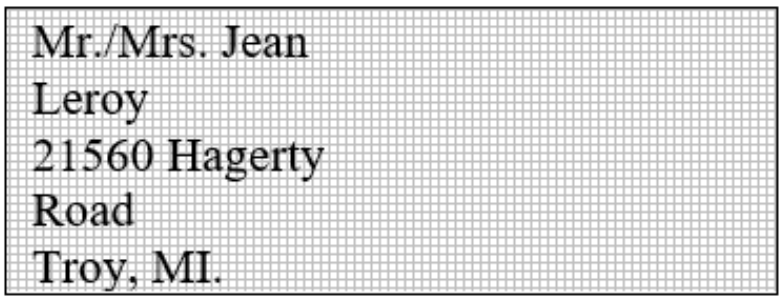
The form:

```
XFSFORM "Bank Details"
BEGIN
  UNIT INCH, 16, 16
  SIZE 121, 64
  VERSION 1, 0, "13/09/96", "XFS Editor"
  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

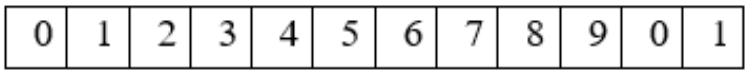
The form:

```
XFSFORM "Smart Account Number"
BEGIN
  UNIT INCH, 16, 16
  SIZE 121, 64
  VERSION 1, 0, "13/09/96", "XFS Editor"
  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:

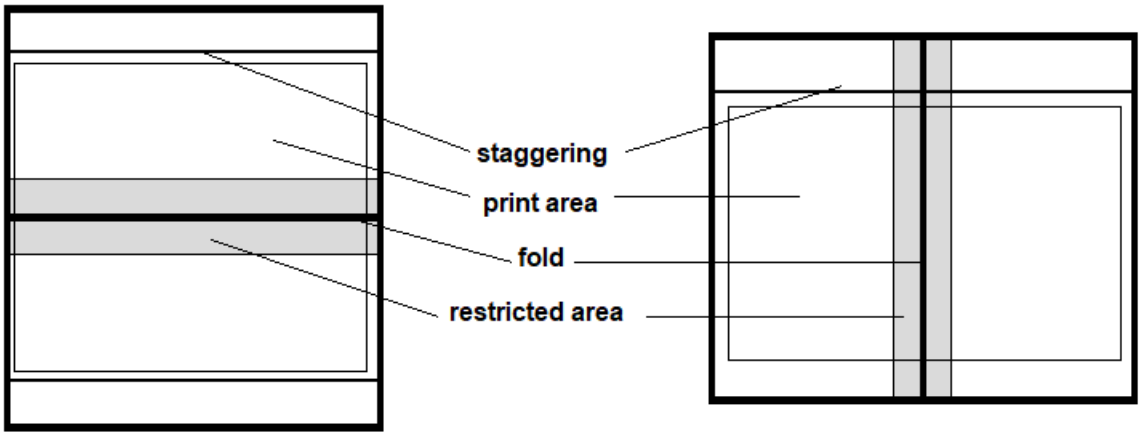


Media Definition

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 services responsibility to ensure that the form definition does not specify the printing of any fields that conflict with the media definition. An example of such a conflict might be that the form definition asks for a field to be printed in an area that the media definition defines as an unprintable area.

The media definition is also intended to provide the capability of defining media types that are specific to the financial industry. An example is a passbook as shown below.

Passbook with horizontal fold **Passbook with vertical fold**



Keyword	Nested Keyword	Required	Names	Notes
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
	UNIT	✓	*base,x,y*	**base** - Base resolution unit for form definition: **MM**, **INCH** or **ROWCOLUMN** **x** - Horizontal base unit fraction **y** - Vertical base unit fraction
	SIZE	✓	*width,height*	**width** - Width of physical media **height** - Height of physical media (0 = unlimited, i.e., roll paper)
	PRINTAREA		*x,y,width,height*	Printable area relative to top left corner of physical media (default = physical size of media)
	RESTRICTED		*x,y,width,height*	Restricted area relative to top left corner of physical media (default = no restricted area)
	FOLD		*fold*	Type of passbook: **HORIZONTAL** (default) or **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		✓		

Form and Media Definitions in Multi-Vendor Environments

In a multi-vendor environment, the capabilities of the service and hardware may be different, therefore the following should be considered.

- 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.

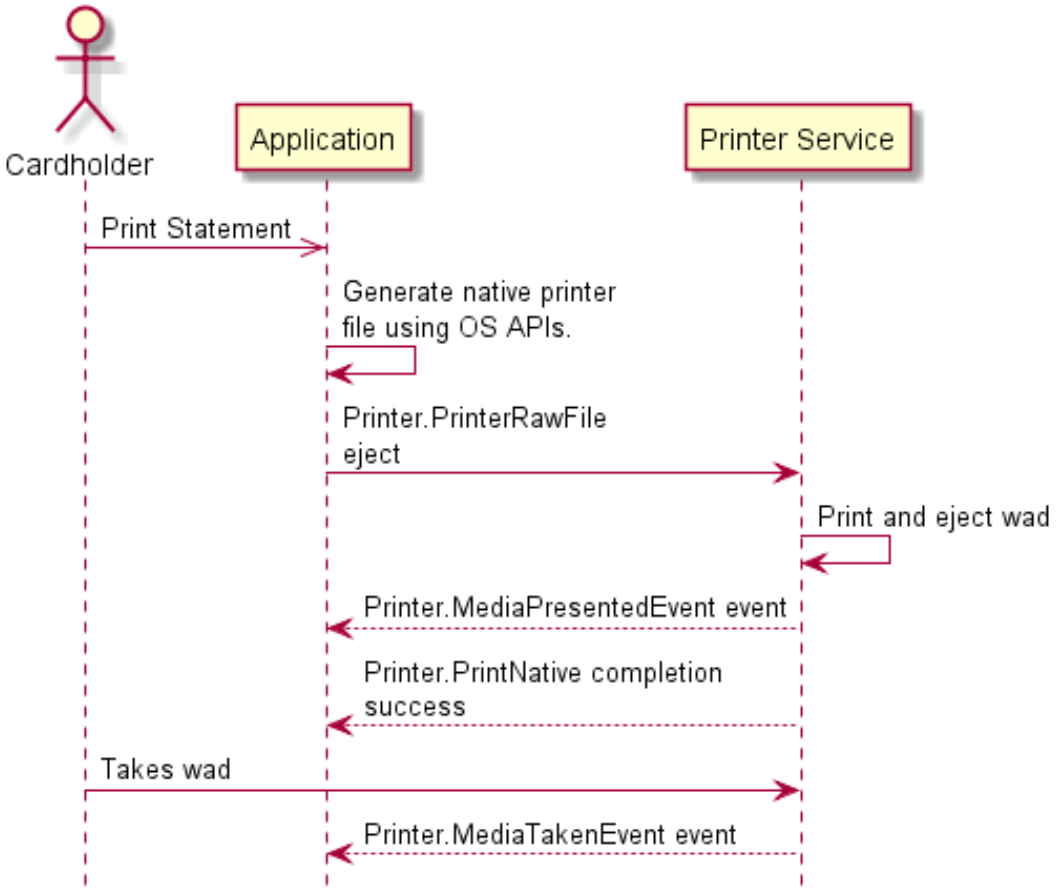
13.1.6 Command and Event Flows during Single and Multi-Page / Wad Printing

It is possible to print a number of pages or bunches of pages (wads) through the Service. 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.PrintNative, Printer.PrintForm and Printer.PrintRaw) where the following conditions apply:

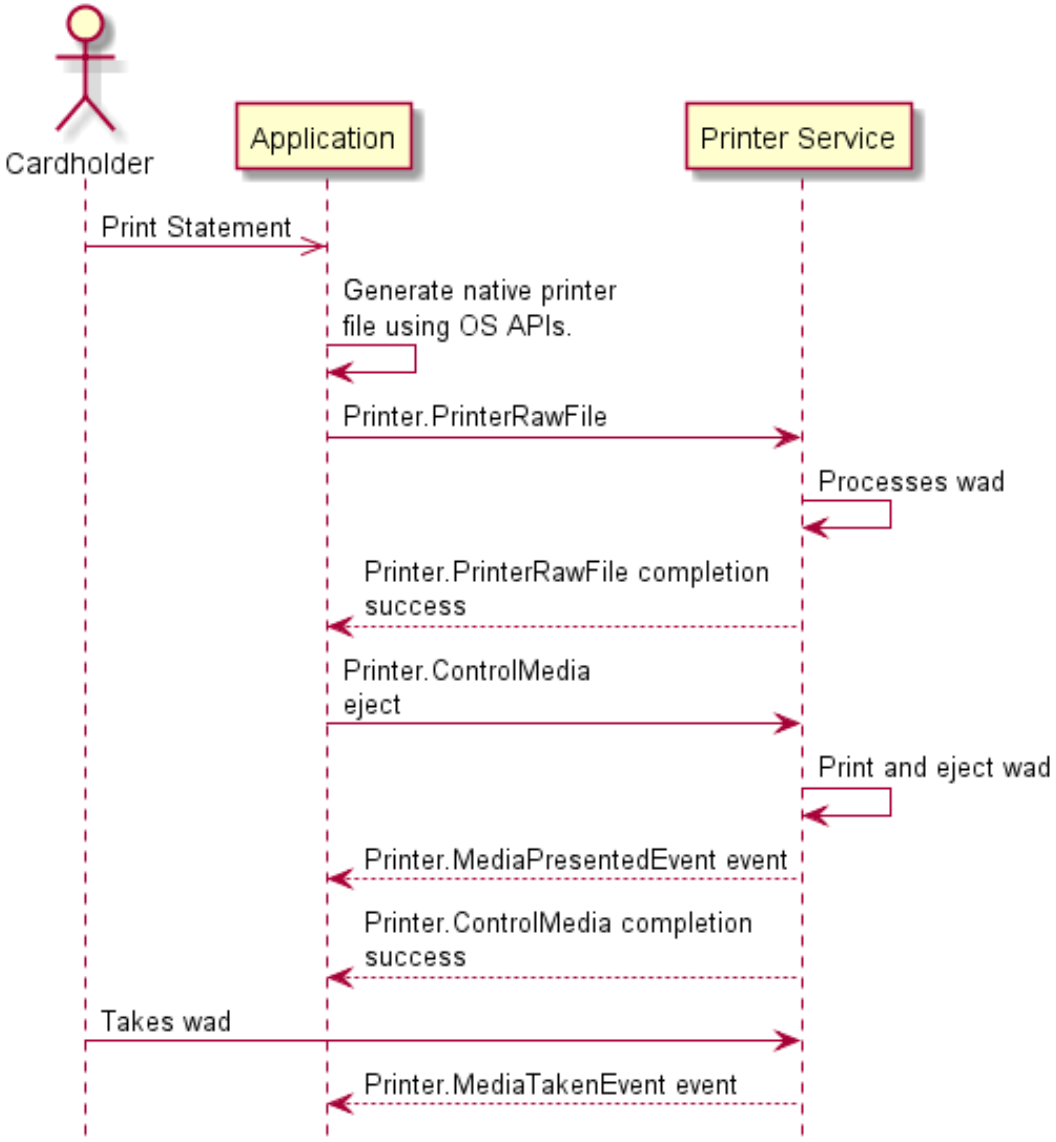
- A single page or single wad of pages is presented.
- The mediaPresented capability is true (indicates that the Printer.MediaPresentedEvent event can be generated).
- The mediaControl in the command data is set to *eject*. The Printer.PrintNative command is used as an example.



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.PrintNative, Printer.PrintForm and Printer.PrintRaw) where the following conditions apply:

- A single page or single wad of pages is presented.
- The mediaPresented is true (indicates that the Printer.MediaPresentedEvent event can be generated).
- The mediaControl is not included in the command data. The Printer.PrintNative command is used as an example.
- The media is presented to the user through a Printer.ControlMedia command, with the mediaControl property set to *eject*.

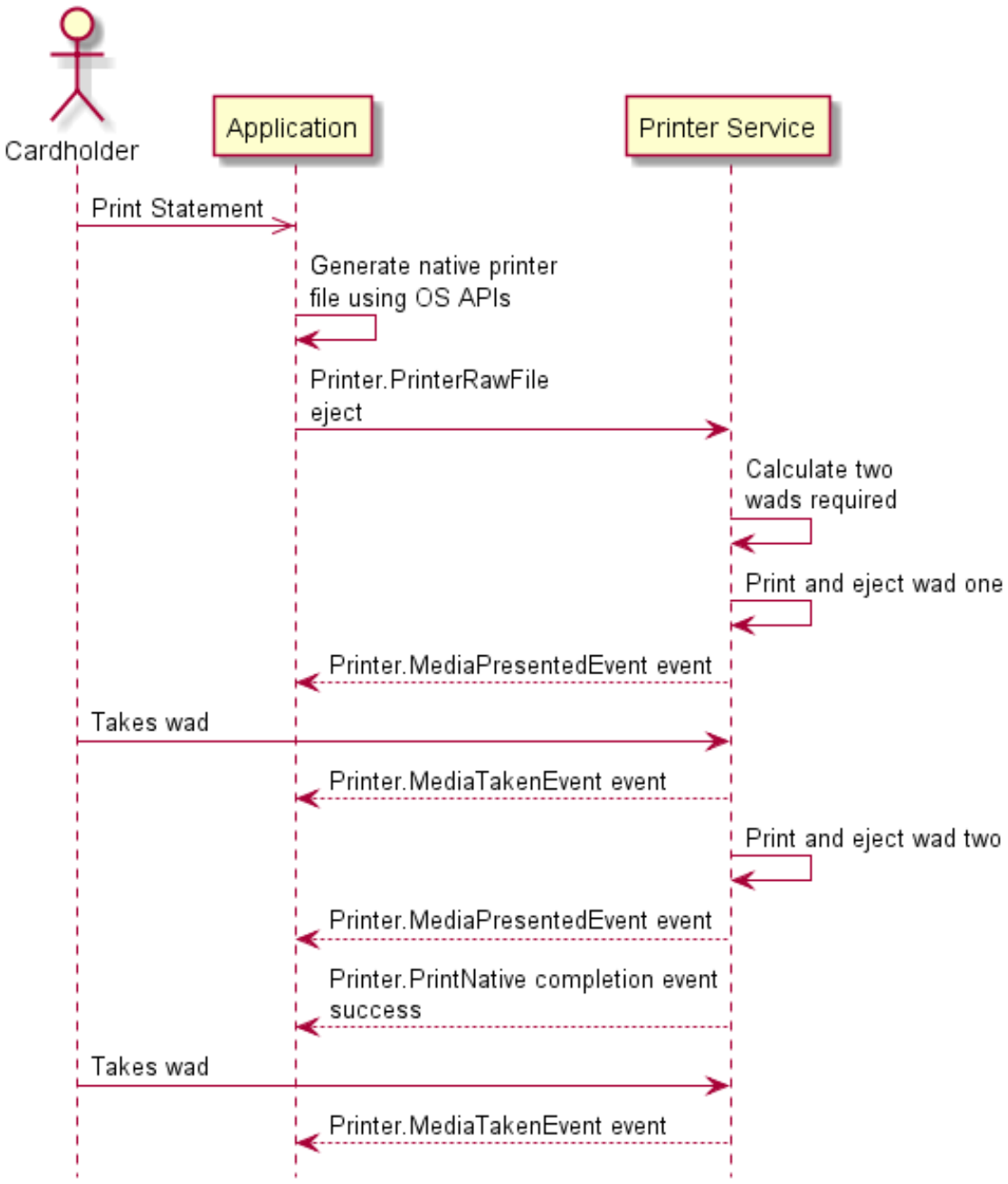


Multi Page / Multi Wad Printing With Immediate Media Control

This diagram illustrates a successful Printer.PrintNative 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 is set to *eject*.
- The previous page/wad must be removed before subsequent pages/wads can be presented.

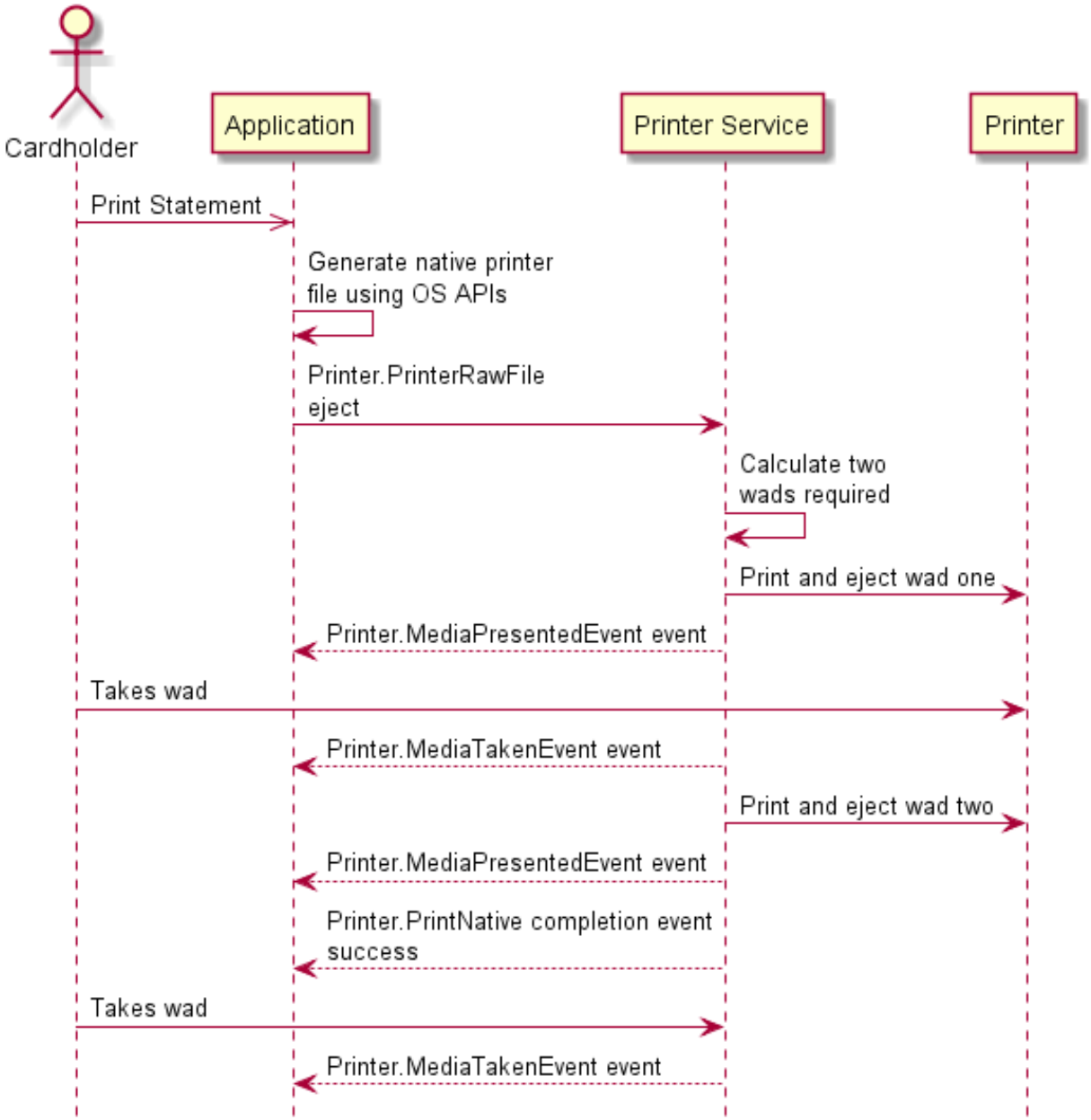


Multi Page / Multi Wad Printing With Separate Media Control

This diagram illustrates a successful Printer.PrintNative 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 property is omitted.
- The media is presented to the user through a Printer.ControlMedia command, with the mediaControl property set to *eject*.
- The previous page/wad must be removed before subsequent pages/wads can be presented.


Cardholder
Application
Printer Service
Printer
Print Statement
Generate native printer file using OS APIs
Printer.PrinterRawFile eject
Calculate two wads required
Print and eject wad one
Printer.MediaPresentedEvent event
Takes wad
Printer.MediaTakenEvent event
Print and eject wad two
Printer.MediaPresentedEvent event
Printer.PrintNative completion event success
Takes wad
Printer.MediaTakenEvent event

13.2 Command Messages

13.2.1 Printer.GetFormList

This command is used to retrieve the list of forms available on the device.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"formList": ["Form1", "Form2"]	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

13.2.2 Printer.GetMediaList

This command is used to retrieve the list of media definitions available on the device.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"mediaList": ["Media1", "Media2"]	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

13.2.3 Printer.GetQueryForm

This command is used to retrieve details of the definition of a specified form.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"formName": "Add example"	string	
}		

Properties

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

formName
The form name for which to retrieve details.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "formNotFound",	string	
"formName": "Form 1",	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": "User prompt1",	string	
"fields": ["Field1, "Field2"]	array (string)	
}		

Properties

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

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 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. Property value constraints: `minimum: 0`
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. Property value constraints: `minimum: 0`
width Specifies the width of the form in terms of the base horizontal resolution. Property value constraints: `minimum: 0`
height Specifies the height of the form in terms of the base vertical resolution. Property value constraints: `minimum: 0`
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. Property value constraints: `minimum: 0`

Properties
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. Property value constraints: `minimum: 0`
versionMajor Specifies the major version of the form. Omitted if the version is not specified in the form. Property value constraints: `minimum: 0`
versionMinor Specifies the minor version of the form. Omitted if the version is not specified in the form. Property value constraints: `minimum: 0`
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

13.2.4 Printer.GetQueryMedia

This command is used to retrieve details of the definition of a specified media.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"mediaName": "Add example"	string	
}		

Properties

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

mediaName
The media name for which to retrieve details.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "mediaNotFound",	string	
"mediaType": "generic",	string	
"base": "inch",	string	
"unitX": 0,	integer	
"unitY": 0,	integer	
"sizeWidth": 0,	integer	
"sizeHeight": 0,	integer	
"pageCount": 0,	integer	
"lineCount": 0,	integer	
"printAreaX": 0,	integer	
"printAreaY": 0,	integer	
"printAreaWidth": 0,	integer	
"printAreaHeight": 0,	integer	
"restrictedAreaX": 0,	integer	
"restrictedAreaY": 0,	integer	
"restrictedAreaWidth": 0,	integer	
"restrictedAreaHeight": 0,	integer	
"stagger": 0,	integer	
"foldType": "none",	string	
"paperSources": {	object	

Payload (version 1.0)	Type	Required
"upper": false,	boolean	
"lower": false,	boolean	
"external": false,	boolean	
"aux": false,	boolean	
"aux2": false,	boolean	
"park": false,	boolean	
"exampleProperty1": false,	boolean	
"exampleProperty2": 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:

- `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 generic, i.e. a single sheet.
- `passbook` - The media is a passbook.
- `multipart` - The media is a multi-part.

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.

Property value constraints:

```
minimum: 0
```

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.

Property value constraints:

```
minimum: 0
```

sizeWidth

Specifies the width of the media in terms of the base horizontal resolution.

Property value constraints:

```
minimum: 0
```

Properties
sizeHeight Specifies the height of the media in terms of the base vertical resolution. Property value constraints: `minimum: 0`
pageCount Specifies the number of pages in a media of type *passbook*. Property value constraints: `minimum: 0`
lineCount Specifies the number of lines on a page for a media of type *passbook*. Property value constraints: `minimum: 0`
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. Property value constraints: `minimum: 0`
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. Property value constraints: `minimum: 0`
printAreaWidth Specifies the printable area width of the media in terms of the base horizontal resolution. Property value constraints: `minimum: 0`
printAreaHeight Specifies the printable area height of the media in terms of the base vertical resolution. Property value constraints: `minimum: 0`
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. Property value constraints: `minimum: 0`
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. Property value constraints: `minimum: 0`
restrictedAreaWidth Specifies the restricted area width of the media in terms of the base horizontal resolution. Property value constraints: `minimum: 0`
restrictedAreaHeight Specifies the restricted area height of the media in terms of the base vertical resolution. Property value constraints: `minimum: 0`

Properties
stagger Specifies the staggering from the top in terms of the base vertical resolution for a media of type *passbook*. Property value constraints: `minimum: 0`
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. If omitted, the paper source is determined by the Service.
paperSources/upper The upper paper source.
paperSources/lower The lower paper source.
paperSources/external The external paper source.
paperSources/aux The auxiliary paper source.
paperSources/aux2 The second auxiliary paper source.
paperSources/park The parking station.
paperSources/exampleProperty1 (example name) The vendor specific paper source. Property name constraints: `pattern: ^[a-zA-Z]([a-zA-Z0-9]*)$`

Event Messages

None

13.2.5 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"formName": "Add example",	string	
"fieldName": "Add example"	string	
}		

Properties

timeout
Timeout in milliseconds for the command to complete. If set to 0, 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 (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "formNotFound",	string	
"fields": {	object	
"*exampleProperty1*": {	object	
"indexCount": 0,	integer	
"type": "text",	string	
"class": "static",	string	
"access": "read",	string	
"overflow": "terminate",	string	
"initialValue": "This is Field 1",	string	
"format": "Format 1",	string	
"coercivity": "auto"	string	
},		
"*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. • `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 0 indicates that this field is not an index field. Index fields are typically used to present information in a tabular fashion. Property value constraints: `minimum: 0`
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.
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.

Properties
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 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

13.2.6 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"codelineFormat": "cmc7"	string	
}		

Properties

timeout
Timeout in milliseconds for the command to complete. If set to 0, 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 property can be one of the following values:

- `cmc7` - Report the CMC7 mapping.
- `e13b` - Report the E13B mapping.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"codelineFormat": "cmc7",	string	
"charMapping": "Add example"	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.

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

13.2.7 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 (version 1.0)	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 0, the command will not timeout but can be canceled.

default: 0

mediaControl

Specifies the manner in which the media should be handled, as a combination of the following properties:

It is not possible to combine the properties eject, retract, park, expel and ejectToTransport with each other otherwise the command completes with *invalidData*.

It is not possible to combine the property clearBuffer with any other properties, otherwise the command completes with *invalidData*.

An application should be aware that the sequence of the actions is not guaranteed if more than one property is specified in this parameter.

Properties
mediaControl/eject Flush any data to the printer that has not yet been printed from previous Printer.PrintForm or Printer.PrintNative 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.PrintNative 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.
mediaControl/expel Flush the data as per flush, then throw the media out of the exit slot.
mediaControl/ejectToTransport Flush the data as per flush, then move the media to a position on the transport just behind the exit slot.
mediaControl/rotate180 Flush the data as per flush, then rotate media 180 degrees in the printing plane.
mediaControl/clearBuffer Clear any data that has not yet been physically printed from previous Printer.PrintForm or Printer.PrintNative commands.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "noMediaPresent"	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:

- `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.MediaPresentedEvent

13.2.8 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*.
- If the data overflows the field and the field definition OVERFLOW value is TRUNCATE, BESTFIT, OVERWRITE or WORDWRAP, a *Printer.FieldWarningEvent* is posted with *failure overflow*.
- Other field-related problems generate a *fieldError* error code and a Printer.FieldErrorEvent.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"formName": "Form1",	string	
"mediaName": "Media1",	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	
"clearBuffer": false	boolean	
},		
"fields": {	object	
"*exampleProperty1*": "Add example",	string	

Payload (version 1.0)	Type	Required
`"exampleProperty2": "Add example"`	string	
`},`		
`"paperSource": "Add example"`	string	
`}`		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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). If not specified, 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). If not specified, the *yoffset* value from the form definition should be used.

Property value constraints:

```
minimum: 0
```

resolution

Specifies the resolution in which to print the form. Possible values are:

- `low` - Print form with low resolution.
- `medium` - Print form with medium resolution.
- `high` - Print form with high resolution.
- `veryHigh` - Print form with very high resolution.

mediaControl

Specifies the manner in which the media should be handled after the printing is done. If no options are set, it means do none of these actions, as when printing multiple forms on a single page. When no options 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 option 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.PrintNative commands, then eject the media.

Properties
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.PrintNative 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.
mediaControl/expel Flush the data as per flush, then throw the media out of the exit slot.
mediaControl/ejectToTransport Flush the data as per flush, then move the media to a position on the transport just behind the exit slot.
mediaControl/rotate180 Flush the data as per flush, then rotate media 180 degrees in the printing plane.
mediaControl/clearBuffer Clear any data that has not yet been physically printed from previous Printer.PrintForm or Printer.PrintNative 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.
fields/exampleProperty1 (example name)

Properties
paperSource Specifies the paper source to use when printing this form. If omitted, then the paper source is determined from the media definition. This parameter is ignored if there is already paper in the print position. It can be one of the following: • `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. • `aux` - Use the auxiliary paper source. • `aux2` - Use the second auxiliary paper source. • `park` - Use the parking station paper source. • `<paper source identifier>` - The vendor specific paper source. Property value constraints: `pattern: ^upper$\|^lower$|^external$\|^aux$|^aux2$\|^park$|^[a-zA-Z]([a-zA-Z0-9]*)$`

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	string	
`"errorCode": "formNotFound"`	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.

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 the Service 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.MediaPresentedEvent
- Printer.MediaRejectedEvent

13.2.9 Printer.PrintRaw

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 may not be aware of).
2. Usage of this command will not be portable.
3. This command violates the XFS4IoT forms model that is the basis of XFS4IoT printer access.

Thus usage of this command should be avoided whenever possible.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"inputData": "no",	string	
"data": "UmF3RGF0YQ=="	string	
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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.

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "shutterFail",	string	
"data": "UmF3RGF0YQ=="	string	
}		

Properties

completionCode

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

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: • `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.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

Event Messages

- Printer.MediaPresentedEvent

13.2.10 Printer.PrintNative

This command is used to print data using the native printer language. The creation and content of this data 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"data": "UmF3RGF0YQ==",	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": "lower"	string	
}		

Properties

timeout

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

default: 0

Properties

data
The data to be printed.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

mediaControl Specifies the manner in which the media should be handled after each page is printed. If no options are set, no actions will be performed, as when printing multiple pages on a single media item. Note that the clearBuffer option 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.PrintNative 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.PrintNative 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.
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.

Properties
mediaControl/rotate180 Flush the data as per flush, then rotate media 180 degrees in the printing plane.
mediaControl/clearBuffer Clear any data that has not yet been physically printed from previous Printer.PrintForm or Printer.PrintNative commands.
paperSource Specifies the paper source to use when printing. If omitted, the Service will determine the paper source that will be used. This parameter is ignored if there is already paper in the print position. It can be one of the following: • `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. • `aux` - Use the auxiliary paper source. • `aux2` - Use the second auxiliary paper source. • `park` - Use the parking station paper source. • `<paper source identifier>` - The vendor specific paper source. Property value constraints: `pattern: ^upper$\|^lower$|^external$\|^aux$|^aux2$\|^park$|^[a-zA-Z]([a-zA-Z0-9]*)$`

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	string	
`"errorCode": "shutterFail"`	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.

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. • `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. • `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

13.2.11 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 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 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"formName": "Form1",	string	
"fieldNames": ["FieldName1"],	array (string)	
"mediaName": "MediaName1",	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	

Payload (version 1.0)	Type	Required
"rotate180": false,	boolean	
"clearBuffer": false	boolean	
}		
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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. The clearBuffer option 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.PrintNative 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.PrintNative 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.

Properties
mediaControl/backward Flush the data as per flush, then turn one page backward.
mediaControl/turnMedia Flush the data as per flush, then turn inserted media.
mediaControl/stamp Flush the data as per flush, then stamp on inserted media.
mediaControl/park Park the media in the parking station.
mediaControl/expel Flush the data as per flush, then throw the media out of the exit slot.
mediaControl/ejectToTransport Flush the data as per flush, then move the media to a position on the transport just behind the exit slot.
mediaControl/rotate180 Flush the data as per flush, then rotate media 180 degrees in the printing plane.
mediaControl/clearBuffer Clear any data that has not yet been physically printed from previous Printer.PrintForm or Printer.PrintNative commands.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "formNotFound",	string	
"fields": {	object	
"*exampleProperty1*": "Add example",	string	
"*exampleProperty2*": "Add example"	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.

Properties
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.MediaRejectedEvent

13.2.12 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), 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.

Command Message

Payload (version 1.0)	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	
}		
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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.

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.

Properties
frontImageColorFormat Specifies the color format of the requested front image as one of the following: • `binary` - The scanned image has to be returned in binary (image contains two colors, usually the colors black and white). • `grayscale` - The scanned image has to be returned in gray scale (image contains multiple gray colors). • `fullcolor` - The scanned image has to be returned in full color (image contains colors like red, green, blue, etc.).
backImageColorFormat Specifies the color format of the requested back image as one of the following: • `binary` - The scanned image has to be returned in binary (image contains two colors, usually the colors black and white). • `grayscale` - The scanned image has to be returned in gray scale (image contains multiple gray colors). • `fullcolor` - The scanned image 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 options (not applicable if no imageSource selected): • `cmc7` - Read CMC7 code line. • `e13b` - Read E13B code line. • `ocr` - Read code line using OCR.
imageSource Specifies the source.
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.

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	string	
`"errorCode": "shutterFail",`	string	
`"images": {`	object	
`"front": {`	object	
`"status": "ok",`	string	
`"data": "SKHFFHGOWORIUNNNLSSL ..."`	string	
`},`		
`"back": {`	object	
`See front properties.`		
`},`		
`"codeline": {`	object	
`See front properties.`		

Payload (version 1.0)	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:

- `shutterFail` - Open or close of the shutter failed due to manipulation or hardware error.
- `mediaJammed` - The media is jammed; operator intervention is required.
- `lampInoperative` - Imaging lamp is inoperative.
- `mediaSize` - The media entered has an incorrect size and the media remains inside the device.
- `mediaRejected` - The media was rejected during the insertion phase. The Printer.MediaRejectedEvent event is posted with the details. The device is still operational.

images

The status and data for each of the requested images.

images/front

The front image status and data.

images/front/status

Status of data source. Possible values are:

- `ok` - The data is OK.
- `notSupported` - 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 a front or back image, this contains 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 printer-1. If the code line was read using the OCR-B font then the ASCII codes will conform to Figure C2 in printer-2. 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.GetCodelineMapping command for the symbols that are unique to MICR fonts.

Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

images/back

The back image status and data.

images/codeline

The codeline status and data.

Event Messages

- Printer.NoMediaEvent
- Printer.MediaInsertedEvent
- Printer.MediaRejectedEvent

13.2.13 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 (version 1.0)	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 0, 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.

Property value constraints:

```
minimum: 0
```

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.

Property value constraints:

```
minimum: 0
```

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "extentNotSupported",	string	
"sizeX": 0,	integer	
"sizeY": 0	integer	
}		

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: • `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. Property value constraints: `minimum: 0`
sizeY Specifies the height of the media in terms of the base vertical resolution. Property value constraints: `minimum: 0`

Event Messages

- Printer.NoMediaEvent
- Printer.MediaInsertedEvent
- Printer.MediaRejectedEvent

13.2.14 Printer.ResetCount

This function resets the present value for number of media items retracted to 0. The function is possible only for printers with retractBins.

The number of media items retracted is controlled by the service and can be requested before resetting using the Common.Status command.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"binNumber": 1	integer	
}		

Properties

timeout

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

default: 0

binNumber

The number of the retract bin for which the retract count should be reset to 0. If omitted or 0, all bin counts will be set to 0. See retractBins.

Property value constraints:

```
minimum: 0
```

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

Properties

completionCode

The completion code.

errorDescription

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

Event Messages

None

13.2.15 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"mediaControl": "eject",	string	
"retractBinNumber": 1	integer	
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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*.

Property value constraints:

```
minimum: 1
```

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "shutterFail"	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.

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.MediaPresentedEvent

13.2.16 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"binNumber": 1	integer	
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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.

Property value constraints:

```
minimum: 1
```

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "noMediaPresent",	string	
"binNumber": 2	integer	
}		

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:

- `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.

Properties
binNumber The number of the retract bin where the media has actually been deposited. Omitted if no media was retracted. See retractBins. Property value constraints: `minimum: 1`

Event Messages

None

13.2.17 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"paperSource": "Add example"	string	
}		

Properties

timeout

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

default: 0

paperSource

Specifes the paper source to be used. This property can be omitted if any paper source to be used and the paper source is determined by the service. It can be one of the following:

- `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.
- `aux` - Use the auxiliary paper source.
- `aux2` - Use the second auxiliary paper source.
- `park` - Use the parking station paper source.
- `<paper source identifier>` - The vendor specific paper source.

Property value constraints:

```
pattern: ^upper$|^lower$|^external$|^aux$|^aux2$|^park$|^[a-zA-Z]([a-zA-Z0-9]*)$
```

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "paperJammed"	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.

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.MediaPresentedEvent

13.2.18 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"definition": "FormDefinition1",	string	
"overwrite": false	boolean	
}		

Properties

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

definition
This 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 included in this property.

overwrite
Specifies if an existing form or media definition with the same name is to be replaced. If 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 false this command will fail with an error if the form or media definition already exists.
default: false

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "formInvalid"	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.

Properties
errorCode Specifies the error code if applicable. The following values are possible: • `formInvalid` - The form is invalid. • `mediaInvalid` - The media definition is invalid. • `definitionExists` - The specified form or media definition already exists and *overwrite* was false.

Event Messages

- Printer.DefinitionLoadedEvent

13.2.19 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 the Service, *unsupportedData* should be returned, and no replenishment actions will be taken by the Service.

Command Message

Payload (version 1.0)	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 0, 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.

ink
The ink supply was replenished.

lamp
The imaging lamp was replaced.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

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

Event Messages

None

13.2.20 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"action": "forward",	string	
"count": 3	integer	
}		

Properties

timeout
Timeout in milliseconds for the command to complete. If set to 0, 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. If action is *closeForward* or *closeBackward*, this will be ignored.
Property value constraints:

```
minimum: 1
```

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "noMediaPresent"	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.

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

13.2.21 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"blackMarkMode": false	boolean	
}		

Properties

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

blackMarkMode
Specifies whether black mark detection and associated functionality is enabled.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

Properties

completionCode
The completion code.

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

Event Messages

None

13.3 Event Messages

13.3.1 Printer.MediaPresentedEvent

This event is used to indicate when media has been presented to the customer for removal.

Event Message

Payload (version 1.0)	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, 0 if not known.

Property value constraints:

```
minimum: 0
```

13.3.2 Printer.NoMediaEvent

This event specifies that the physical media must be inserted into the device in order for the command to proceed.

Event Message

Payload (version 1.0)	Type	Required
{		
"userPrompt": "Enter paper"	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 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.

13.3.3 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 prompt from the screen telling the user to insert media.

Event Message

Payload (version 1.0)	Type	Required
{		
}		

13.3.4 Printer.FieldErrorEvent

This event specifies that a fatal error has occurred while processing a field.

Event Message

Payload (version 1.0)	Type	Required
{		
"formName": "Form1",	string	
"fieldName": "Field1",	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.

13.3.5 Printer.FieldWarningEvent

This event specifies that a non-fatal error has occurred while processing a field.

Event Message

Payload (version 1.0)	Type	Required
{		
"formName": "Form1",	string	
"fieldName": "Field1",	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.

13.3.6 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.

Event Message

Payload (version 1.0)	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.

13.4 Unsolicited Messages

13.4.1 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.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
}		

13.4.2 Printer.MediaInsertedUnsolicitedEvent

This event specifies that the physical media has been inserted into the device without any read or print commands being executed. This event is only generated when media is entered in an unsolicited manner.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
}		

13.4.3 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 XFS4IoT interface.

Unsolicited Message

Payload (version 1.0)	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, 0 if not known.

Property value constraints:

```
minimum: 0
```

13.4.4 Printer.MediaDetectedEvent

This event is generated when a media is detected in the device during a reset operation.

Unsolicited Message

Payload (version 1.0)	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*.

13.4.5 Printer.RetractBinStatusEvent

This event specifies that the status of the retract bin holding the retracted media has changed.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
"binNumber": 2,	integer	
"state": "inserted"	string	
}		

Properties

binNumber

Number of the retract bin for which the status has changed.

Property value constraints:

```
minimum: 1
```

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.

13.4.6 Printer.DefinitionLoadedEvent

This event is used to indicate when a form or media definition has successfully been loaded via the Printer.LoadDefinition command.

Unsolicited Message

Payload (version 1.0)	Type	Required
`{`		
`"name": "form name",`	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.

13.4.7 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.

Unsolicited Message

Payload (version 1.0)	Type	Required
`{`		
`"retractResult": "ok",`	string	
`"binNumber": 2`	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 0 if the media is retracted to the transport or retractResult is *jammed*.

Property value constraints:

```
minimum: 0
```

13.4.8 Printer.RetractBinThresholdEvent

This event specifies that the status of the retract bin holding the retracted media has changed.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
"binNumber": 2,	integer	
"state": "ok"	string	
}		

Properties

binNumber

Number of the retract bin for which the status has changed.

Property value constraints:

```
minimum: 1
```

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.

13.4.9 Printer.PaperThresholdEvent

This 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.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
"paperSource": "Add example",	string	
"threshold": "full"	string	
}		

Properties

paperSource

Specifies the paper source as one of the following:

- `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.
- `aux` - Use the auxiliary paper source.
- `aux2` - Use the second auxiliary paper source.
- `park` - Use the parking station paper source.
- `<paper source identifier>` - The vendor specific paper source.

Property value constraints:

```
pattern: ^upper$|^lower$|^external$|^aux$|^aux2$|^park$|^[a-zA-Z]([a-zA-Z0-9]*)$
```

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.

13.4.10 Printer.TonerThresholdEvent

This event is used to specify that the state of the toner (or ink) reached a threshold.

Unsolicited Message

Payload (version 1.0)	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.

13.4.11 Printer.LampThresholdEvent

This event is used to specify that the state of the imaging lamp reached a threshold.

Unsolicited Message

Payload (version 1.0)	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.

13.4.12 Printer.InkThresholdEvent

This event is used to specify that the state of the stamping ink reached a threshold.

Unsolicited Message

Payload (version 1.0)	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.

14. Text Terminal Interface

This chapter defines the Text Terminal interface functionality and messages.

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.

14.1 General Information

14.1.1 References

ID	Description
textterminal-1	ISO/IEC 646 (ASCII)

14.1.2 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 and font names are case sensitive.
- **Strings**
  All strings must be enclosed in double quote characters ("). Standard C escape sequences are allowed.
- **Comments**
  Start with two forward slashes (//); end at line termination.

Other notes:

- If a keyword is present, all its values must be specified; default values are used only if the keyword is absent.
- Values that are character strings are marked with * in the definitions and must be quoted as specified above.
- Fields are processed in the sequence they are defined in the form.
- The order of attributes within a form is not mandatory; the attributes may be defined in any order.
- All definitions must be encoded in UTF-8. Keywords are restricted to an internal representation of ISO 646 [Ref. textterminal-1]] (ANSI) characters.

XFS form/media definition in multi-vendor environments

In a multi-vendor environment, the capabilities of the service and hardware may be different, therefore the following should be considered.

- Physical display area dimensions may vary from one text terminal to another.
- Some form/media definition keywords may not be supported due to limitations of the hardware or software.

Form Definition

Keyword	Nested Keyword	Required	Names	Notes
FORM		✓	*formname**	
BEGIN		✓		
	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
	COPYRIGHT		*copyright**	Copyright entry
	TITLE		*title**	Title of form
	COMMENT		*comment**	Comment section
	[XFSFIELD **BEGIN** ... **END]**		*fieldname**	One field definition for each field in the form
END		✓		

Field Definition

Keyword	Nested Keyword	Required	Names	Notes
FIELD		✓	*fieldname**	
BEGIN		✓		
	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)
	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)

Keyword	Nested Keyword	Required	Names	Notes
	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 '|' operator: - **NORMAL** (default) - **UNDER** (single underline) - **INVERTED** - **FLASHING**
	HORIZONTAL		*justify*	Horizontal alignment of field contents: - **LEFT** (default) - **RIGHT** - **CENTER**
	FORMAT		*formatstring**	This is an application defined input field describing how the application should format the data. This may be interpreted by the Service. For *GRAPHIC* fields, this defines the type of the graphic, for example, "BMP", "PNG" etc.
	INITIALVALUE		*value**	Initial value. For *GRAPHIC* fields, this value may contain Base64 encoded image data The type of this graphic will be determined by the FORMAT field.
END		✓		

14.2 Command Messages

14.2.1 TextTerminal.GetFormList

This command is used to retrieve the list of forms available on the device.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"formList": ["Example form1", "Example form2"]	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

14.2.2 TextTerminal.GetQueryForm

This command is used to retrieve details of the definition of a specified form.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"formName": "Example form"	string	Yes
}		

Properties

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

formName
Contains the form name on which to retrieve details.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "formNotFound",	string	
"formName": "Example form",	string	
"width": 40,	integer	
"height": 20,	integer	
"versionMajor": 2,	integer	
"versionMinor": 1,	integer	
"fields": ["Field1", "Field2"]	array (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:

- `formNotFound` - The specified form cannot be found.
- `formInvalid` - The specified form is invalid.

formName
Specifies the name of the form.

Properties
width Specifies the width of the form in columns. Property value constraints: `minimum: 0`
height Specifies the height of the form in rows. Property value constraints: `minimum: 0`
versionMajor Specifies the major version. Omitted if the version is not specified in the form. Property value constraints: `minimum: 0`
versionMinor Specifies the minor version. Omitted if the version is not specified in the form. Property value constraints: `minimum: 0`
fields A list of the field names.

Event Messages

None

14.2.3 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"formName": "My form name",	string	Yes
"fieldName": "My form field"	string	
}		

Properties

timeout
Timeout in milliseconds for the command to complete. If set to 0, 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 omitted, then retrieve details for all fields on the form.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "formNotFound",	string	
"fields": {	object	
"*Field 1*": {	object	
"type": "text",	string	
"class": "static",	string	
"access": {	object	
"read": false,	boolean	
"write": false	boolean	
},		
"overflow": "terminate",	string	
"format": "Format 1"	string	
},		
"*Field 2*": {	object	
See Field 1 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/Field 1 (example name)
fields/Field 1/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/Field 1/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/Field 1/access Specifies whether the field is to be used for input, output or both.
fields/Field 1/access/read The field is used for input from the physical device.
fields/Field 1/access/write The field is used for output to the physical device.
fields/Field 1/overflow Specifies how an overflow of field data should be handled 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.
fields/Field 1/format Format string as defined in the form for this field.

Event Messages

None

14.2.4 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 (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	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|arrowLe
ft|arrowRight|fdk(0[1-9]|[12][0-9]|3[0-2])|oem[A-Za-z0-9]*)$
```

Event Messages

None

14.2.5 TextTerminal.Beep

This command is used to beep at the text terminal unit.

Command Message

Payload (version 1.0)	Type	Required
{		
"beep": {	object	
"continuous": false,	boolean	
"beepType": "keyPress"	string	
},		
"timeout":	undefined	
}		

Properties

beep
Specifies whether the beeper should be turned on or off. If omitted, the beeper is switched off, otherwise the beep is specified as follows.

beep/continuous
Specifies whether the beep is continuous.
default: false

beep/beepType
Specifies the type of beep as one of the following:

- `keyPress` - The beeper sounds a key click signal.
- `exclamation` - The beeper sounds an exclamation signal.
- `warning` - The beeper sounds a warning signal.
- `error` - The beeper sounds an error signal.
- `critical` - The beeper sounds a critical signal.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

Properties

completionCode
The completion code.

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

Event Messages

None

14.2.6 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"positionX": 0,	integer	Yes
"positionY": 0,	integer	Yes
"width": 10,	integer	Yes
"height": 15	integer	Yes
}		

Properties

timeout
Timeout in milliseconds for the command to complete. If set to 0, 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: 1`

height
Specifies the height position of the area to be cleared.
Property value constraints:
`minimum: 1`

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

Properties

completionCode
The completion code.

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

Event Messages

None

14.2.7 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"resolution": {	object	Yes
"sizeX": 0,	integer	
"sizeY": 0	integer	
}		
}		

Properties

timeout

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

default: 0

resolution

This must be one of the supported resolutions.

resolution/sizeX

Specifies 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 (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "resolutionNotSupported"	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.

Properties
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

14.2.8 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"formName": "My form",	string	Yes
"clearScreen": true,	boolean	Yes
"fields": {	object	Yes
"*Field1*": "field",	string	
"*Field2": "field"*	string	
}		
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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.

default: true

fields

Details of the field(s) to write. The property is the field name and value is field value containing all the printable characters (numeric and alphanumeric) to display on the text terminal unit key pad for this field. An example shows two fields to be written.

```
{
  "Field1": 123,
  "Field2": 456
}
```

fields/Field1 (example name)

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "formNotFound"	string	
}		

Properties

completionCode

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

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 *fields* is invalid. • `characterSetsData` - The character set(s) supported by the Service is inconsistent with *fields*. • `fieldError` - An error occurred while processing a field.

Event Messages

- TextTerminal.FieldErrorEvent
- TextTerminal.FieldWarningEvent

14.2.9 TextTerminal.ReadForm

This command is used to read data from input fields on the specified form.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"formName": "My form",	string	Yes
"fields": ["Field1", "Field2"]	array (string)	
}		

Properties

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

formName
Specifies the name of the form.

fields
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 omitted, data is read from all input fields on the form in the order they appear in the form (independent of the field screen position).

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "formNotFound",	string	
"fields": {	object	
"*Field1*": "123",	string	
"*Field2*": "123"	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.

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 *fields* 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. Each property's name is the field name and value is field value containing all the printable characters (numeric and alphanumeric) read from the text terminal unit key pad for this field. An example shows two fields read.
`{`
`"Field1": 123,`
`"Field2": 456`
`}`

fields/Field1 (example name)

Event Messages

- TextTerminal.FieldErrorEvent
- TextTerminal.FieldWarningEvent

14.2.10 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"mode": "relative",	string	Yes
"posX": 0,	integer	Yes
"posY": 0,	integer	Yes
"textAttr": {	object	Yes
"underline": false,	boolean	
"inverted": false,	boolean	
"flash": false	boolean	
},		
"text": "Text to display"	string	Yes
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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 *relative*, this specifies a horizontal offset relative to the current cursor position as a 0 based value.

Property value constraints:

```
minimum: 0
```

posY

If mode is set to absolute, this specifies the absolute vertical position. If mode is *relative*, this specifies a vertical offset relative to the current cursor position as a 0 based value.

Property value constraints:

```
minimum: 0
```

textAttr

Specifies the text attributes used for displaying the text. If omitted or none are set to true then the text will be displayed as normal text.

textAttr/underline

The displayed text will be underlined.

default: false

Properties
textAttr/inverted The displayed text will be inverted. default: false
textAttr/flash The displayed text will be flashing. default: false
text Specifies the text that will be displayed.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	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` - The character set(s) supported by the Service is inconsistent with *text*.

Event Messages

None

14.2.11 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, is returned. The Service takes command keys into account when interpreting the data.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"numOfChars": 0,	integer	Yes
"mode": "relative",	string	Yes
"posX": 0,	integer	Yes
"posY": 0,	integer	Yes
"echoMode": "text",	string	
"echoAttr": {	object	
"underline": false,	boolean	
"inverted": false,	boolean	
"flash": false	boolean	
},		
"visible": true,	boolean	
"flush": false,	boolean	Yes
"autoEnd": true,	boolean	
"activeKeys": "12AaBb",	string	Yes
"activeCommandKeys": {	object	
"*enter*": {	object	
"terminate": false	boolean	
},		
"*oem1*": {	object	
See enter properties.		
}		
}		
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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
```

Properties
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 *absolute*, this specifies the absolute horizontal position. If mode is *relative*, this specifies a horizontal offset relative to the current cursor position as a 0 based value. Property value constraints: `minimum: 0`
posY If mode is *absolute*, this specifies the absolute vertical position. If mode is *relative* this specifies a vertical offset relative to the current cursor position as a 0 based value. Property value constraints: `minimum: 0`
echoMode Specifies how the user input is echoed to the screen as one of the following: • `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. default: "text"
echoAttr Specifies the text attributes with which the user input is echoed to the screen. If none of the attributes 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.
visible Specifies whether the cursor is visible. default: true
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 the Service if the maximum number of printable characters as specified with *numOfChars* is entered. default: true
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.

Properties
activeCommandKeys Specifying the command keys which are active during the execution of the command.
activeCommandKeys/enter (example name) 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. • `oem[A-Za-z0-9]*` - A non-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])|oem[A-Za-z0-9]*)$`
activeCommandKeys/enter/terminate The key is a terminate key. default: false

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	string	
`"errorCode": "keyInvalid",`	string	
`"input": "12345"`	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. • `noActiveKeys` - There are no active keys specified.
input Specifies a string containing all the printable characters (numeric and alphanumeric) read from the text terminal unit key pad.

Event Messages

- TextTerminal.KeyEvent

14.2.12 TextTerminal.Reset

Sends a service reset to the Service. 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 (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		
Properties		
timeout Timeout in milliseconds for the command to complete. If set to 0, the command will not timeout but can be canceled. default: 0		

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		
Properties		
completionCode The completion code.		
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.		

Event Messages

None

14.2.13 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"activeKeys": "0123456789ABCabc",	string	Yes
"activeCommandKeys": {	object	
"*enter*": {	object	
"terminate": false	boolean	
},		
"*oem1*": {	object	
See enter properties.		
}		
}		
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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.

activeCommandKeys

Array specifying the command keys which are active during the execution of the next TextTerminal.ReadForm command.

Properties
activeCommandKeys/enter (example name) 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. • `oem[A-Za-z0-9]*` - A non-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])|oem[A-Za-z0-9]*)$`
activeCommandKeys/enter/terminate The key is a terminate key. default: false

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	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.
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. • `noActiveKeys` - There are no active keys specified.

Event Messages

None

14.2.14 TextTerminal.LoadForm

This command is used to load a form definition into the list of available forms. Once a form definition has been loaded through this command it can be used by any of the other form definition processing commands. Form definitions loaded through this command are persistent. When a form definition is loaded a TextTerminal.FormLoadedEvent event is generated to inform applications that a form definition has been added or replaced.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"definition": "See form description",	string	
"overwrite": false	boolean	
}		

Properties

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

definition
This contains the form definition in text format as described in Form and Field Definitions. Only one form definition can be included in this property.

overwrite
Specifies if an existing form definition with the same name is to be replaced. If this is true then an existing form definition with the same name will be replaced, unless the command fails with an error, where the definition will remain unchanged. If this is false this command will fail with an error if the form definition already exists.
default: false

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "formInvalid"	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:

- `formInvalid` - The form is invalid.
- `definitionExists` - The specified form definition already exists and *overwrite* was false.

Event Messages

- TextTerminal.FormLoadedEvent

14.3 Event Messages

14.3.1 TextTerminal.FieldErrorEvent

This event specifies that a fatal error has occurred while processing a field.

Event Message

Payload (version 1.0)	Type	Required
`{`		
`"formName": "Example form",`	string	Yes
`"fieldName": "Field1",`	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.
- `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.
- `typeNotSupported` - The form field type is not supported with device.
- `charSetForm` - Service does not support character set specified in form.

14.3.2 TextTerminal.FieldWarningEvent

This event is used to specify that a non-fatal error has occurred while processing a field.

Event Message

Payload (version 1.0)	Type	Required
{		
}		

14.3.3 TextTerminal.KeyEvent

This event specifies that any active key has been pressed at the Text Terminal device during TextTerminal.Read. 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.

Event Message

Payload (version 1.0)	Type	Required
{		
"key": "0",	string	
"commandKey": "enter"	string	
}		

Properties

key

On a numeric or alphanumeric key press this parameter holds the value of the key pressed. This is omitted when 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 is omitted when no command key was pressed.

Property value constraints:

```
pattern:
^(enter|cancel|clear|backspace|help|doubleZero|tripleZero|arrowUp|arrowDown|arrowLe
ft|arrowRight|fdk(0[1-9]|[12][0-9]|3[0-2])|oem[A-Za-z0-9]*)$
```

14.3.4 TextTerminal.FormLoadedEvent

This event is used to indicate when a form definition has successfully been loaded via the TextTerminal.LoadForm command.

Event Message

Payload (version 1.0)	Type	Required
{		
"name": "Form 1"	string	
}		
Properties		
name Specifies the name of the form just loaded.		

15. Barcode Reader Interface

This chapter defines the Barcode Reader interface functionality and messages.

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.

15.1 Command Messages

15.1.1 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 timeout 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 (version 1.0)	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	
"msi": false,	boolean	
"plessey": false,	boolean	
"std2Of5": false,	boolean	
"std2Of5Iata": false,	boolean	
"pdf417": false,	boolean	

Payload (version 1.0)	Type	Required
"microPdf417": false,	boolean	
"dataMatrix": false,	boolean	
"maxiCode": false,	boolean	
"codeOne": false,	boolean	
"channelCode": false,	boolean	
"telepenOriginal": false,	boolean	
"telepenAim": false,	boolean	
"rss": false,	boolean	
"rssExpanded": false,	boolean	
"rssRestricted": false,	boolean	
"compositeCodeA": false,	boolean	
"compositeCodeB": false,	boolean	
"compositeCodeC": false,	boolean	
"posiCodeA": false,	boolean	
"posiCodeB": false,	boolean	
"triopticCode39": false,	boolean	
"codablockF": false,	boolean	
"code16K": false,	boolean	
"qrCode": false,	boolean	
"aztec": false,	boolean	
"ukPost": false,	boolean	
"planet": false,	boolean	
"postnet": false,	boolean	
"canadianPost": false,	boolean	
"netherlandsPost": false,	boolean	
"australianPost": false,	boolean	
"japanesePost": false,	boolean	
"chinesePost": false,	boolean	
"koreanPost": false	boolean	
}		
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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 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 supports this feature. If the Service does not support this feature then this property is ignored. If all symbologies should be accepted then the *symbologies* property should be omitted.

Properties
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 EAN-13
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
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

Properties
symbologies/code93 CODE 93
symbologies/code128 CODE 128
symbologies/msi MSI
symbologies/plessey PLESSEY
symbologies/std2Of5 STANDARD 2 of 5 (INDUSTRIAL 2 of 5 also)
symbologies/std2Of5Iata STANDARD 2 of 5 (IATA Version)
symbologies/pdf417 PDF-417
symbologies/microPdf417 MICROPDF-417
symbologies/dataMatrix GS1 DataMatrix
symbologies/maxiCode MAXICODE
symbologies/codeOne CODE ONE
symbologies/channelCode CHANNEL CODE
symbologies/telepenOriginal Original TELEPEN
symbologies/telepenAim AIM version of TELEPEN
symbologies/rss GS1 DataBar™
symbologies/rssExpanded Expanded GS1 DataBar™
symbologies/rssRestricted Restricted GS1 DataBar™
symbologies/compositeCodeA Composite Code A Component
symbologies/compositeCodeB Composite Code B Component
symbologies/compositeCodeC Composite Code C Component
symbologies/posiCodeA Posicode Variation A
symbologies/posiCodeB Posicode Variation B

Properties
symbologies/triopticCode39 Trioptic Code 39
symbologies/codablockF Codablock F
symbologies/code16K Code 16K
symbologies/qrCode QR Code
symbologies/aztec Aztec Codes
symbologies/ukPost UK Post
symbologies/planet US Postal Planet
symbologies/postnet US Postal Postnet
symbologies/canadianPost Canadian Post
symbologies/netherlandsPost Netherlands Post
symbologies/australianPost Australian Post
symbologies/japanesePost Japanese Post
symbologies/chinesePost Chinese Post
symbologies/koreanPost Korean Post

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "barcodeInvalid",	string	
"readOutput": [{	array (object)	
"symbology": "ean128",	string	
"barcodeData": "YmFyY29kZSBkYXRh",	string	
"symbologyName": "code39"	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: • `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, one for each barcode scanned during the read operation

readOutput/symbology

Specifies the barcode symbology recognized. This contains one of the following values returned in the symbologies property 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 property will be *symbologyUnknown*.

The following values are possible:

- `ean128` - GS1-128.
- `ean8` - EAN-8.
- `ean8_2` - EAN-8 with 2 digit add-on.
- `ean8_5` - EAN-8 with 5 digit add-on.
- `ean13` - EAN-13.
- `ean13_2` - EAN-13 with 2 digit add-on.
- `ean13_5` - EAN-13 with 5 digit add-on.
- `jan13` - JAN-13.
- `upcA` - UPC-A.
- `upcE0` - UPC-E.
- `upcE0_2` - UPC-E with 2 digit add-on.
- `upcE0_5` - UPC-E with 5 digit add-on.
- `upcE1` - UPC-E with leading 1.
- `upcE1_2` - UPC-E with leading 1and 2 digit add-on.
- `upcE1_5` - UPC-E with leading 1and 5 digit add-on.
- `upcA_2` - UPC-A with2 digit add-on.
- `upcA_5` - UPC-A with 5 digit add-on.
- `codabar` - CODABAR (NW-7).
- `itf` - Interleaved 2 of 5 (ITF).
- `code11` - CODE 11 (USD-8).
- `code39` - CODE 39.
- `code49` - CODE 49.
- `code93` - CODE 93.
- `code128` - CODE 128.
- `msi` - MSI.
- `plessey` - PLESSEY.
- `std2Of5` - STANDARD 2 of 5 (INDUSTRIAL 2 of 5 also).
- `std2Of5Iata` - STANDARD 2 of 5 (IATA Version).
- `pdf417` - PDF-417.
- `microPdf417` - MICROPDF-417.
- `dataMatrix` - GS1 DataMatrix.
- `maxiCode` - MAXICODE.
- `codeOne` - CODE ONE.
- `channelCode` - CHANNEL CODE.
- `telepenOriginal` - Original TELEPEN.
- `telepenAim` - AIM version of TELEPEN.
- `rss` - GS1 DataBar™.
- `rssExpanded` - Expanded GS1 DataBar™.
- `rssRestricted` - Restricted GS1 DataBar™.
- `compositeCodeA` - Composite Code A Component.
- `compositeCodeB` - Composite Code B Component.
- `compositeCodeC` - Composite Code C Component.
- `posiCodeA` - Posicode Variation A.
- `posiCodeB` - Posicode Variation B.
- `trioptic Code39` - Trioptic Code 39.
- `codablockF` - Codablock F.
- `code16K` - Code 16K.
- `qrCode` - QR Code.
- `aztec` - Aztec Codes.

Properties
• `ukPost` - UK Post. • `planet` - US Postal Planet. • `postnet` - US Postal Postnet. • `canadianPost` - Canadian Post. • `netherlandsPost` - Netherlands Post. • `australianPost` - Australian Post. • `japanesePost` - Japanese Post. • `chinesePost` - Chinese Post. • `koreanPost` - Korean Post. • `symbologyUnknown` - The barcode reader was unable to recognize the symbology.

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 property 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.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

readOutput/symbologyName A vendor dependent symbology identifier for the symbology recognized.

Event Messages

None

15.1.2 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 (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		
Properties		
timeout Timeout in milliseconds for the command to complete. If set to 0, the command will not timeout but can be canceled. default: 0		

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		
Properties		
completionCode The completion code.		
errorDescription If included, this contains additional vendor dependent information to assist with problem resolution.		

Event Messages

None

16. Biometric Interface

This chapter defines the Biometric interface functionality and messages.

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 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)**.

16.1 General Information

16.1.1 References

ID	Description
biometric-1	ANSI INCITS 381-2004 Information Technology - Finger Image-Based Data Interchange Format.
biometric-2	ANSI INCITS 378-2004 Information Technology - Finger Minutiae Format for Data Interchange.
biometric-3	ISO/IEC 19794-4:2005 Information technology - Biometric data interchange formats - Part 4: Finger image data.
biometric-4	ISO/IEC 19794-2:2005 Information technology - Biometric data interchange formats - Part 2: Finger minutiae data.

16.1.2 Enrollment

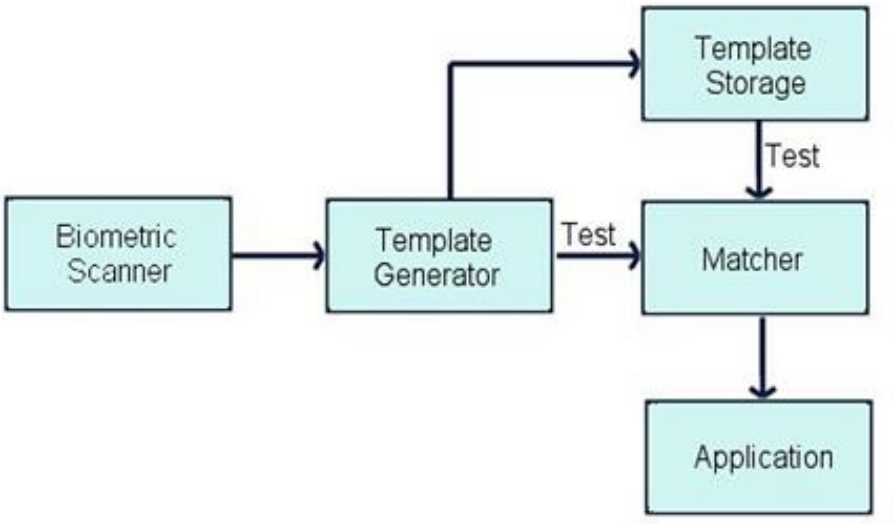
The first time an individual uses a biometric device it is called Enrollment. During enrollment, biometric data from an individual is captured and stored somewhere, for example on a smart card or in a server/host database. Normally the raw biometric data captured will be processed and converted to a 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.

16.1.3 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.



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.

16.1.4 Biometric Device Types

There are many different varieties of biometric hardware, this biometrics specification supports three main different types of devices:

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, but matching is performed externally, for example on a smart card or on a server. In this case the Biometric.Match and Biometric.SetMatch 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. When the Biometric.Read is then called the scanned user data is temporarily stored. The Biometric.Match 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. In this case the Biometric.SetMatch must be called first, either as a one time call or before each Biometric.Read. The purpose of the Biometric.SetMatch is to set the criteria for matching. When the Biometric.Read is then called it scans the user's biometric data and also performs the comparison as a single operation. The Biometric.Match is then called to return the result of the comparison.

16.1.5 Biometric Data Security

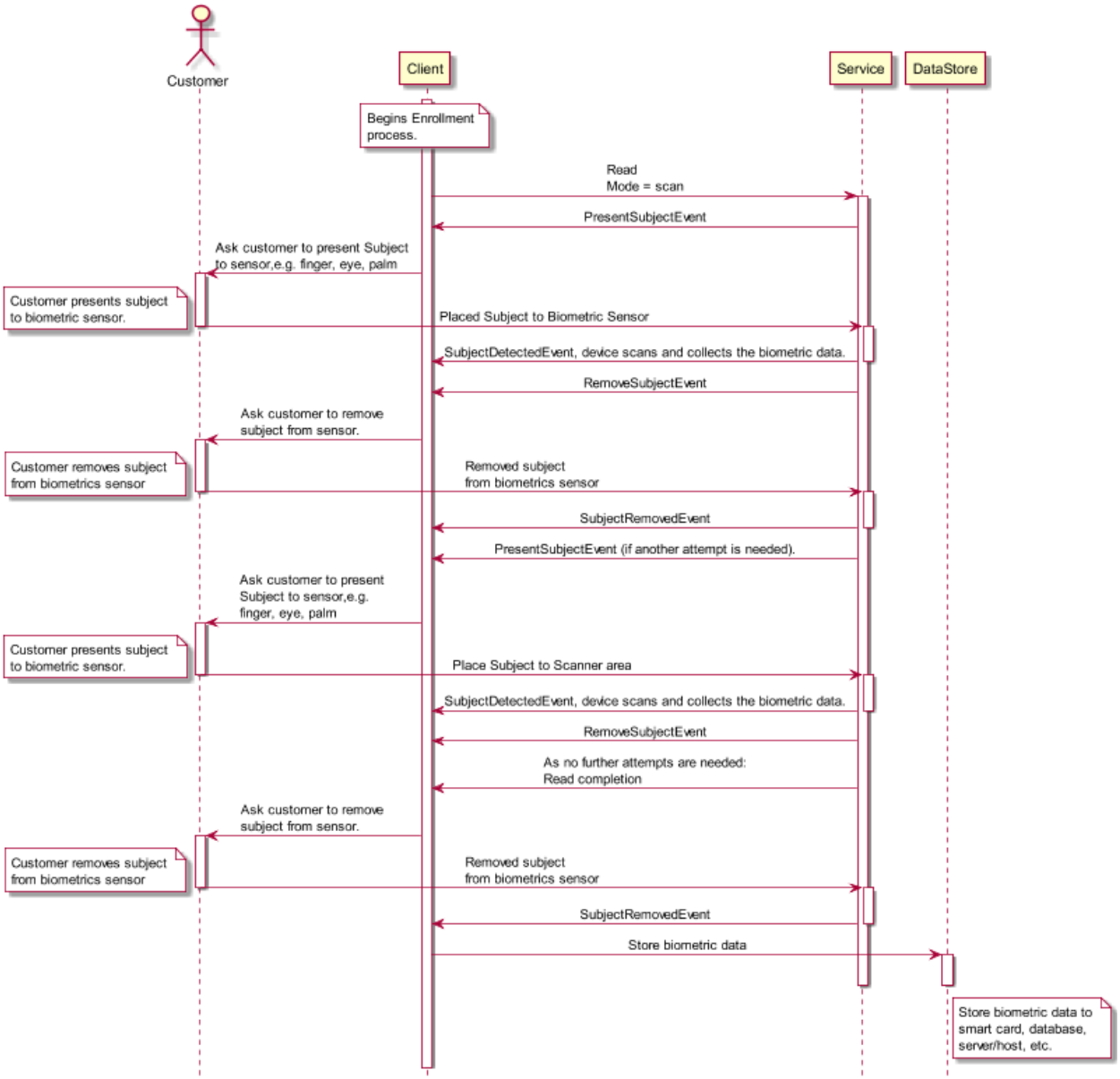
It is recommended that biometric data should be treated with the same strict caution as any other identifying and sensitive information. A well-designed biometric data handling architecture should always be designed to protect against internal tampering, external attacks and other malicious threats. There are various ways of implementing good security of which two 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 the Service. In addition, the KeyManagement interface commands can be used for key management. In this case the Service 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.

16.1.6 Biometric Device Command Flows

Biometric Enrollment Command Flow

The following diagram describes the flow of enrolling a user using the Biometric.Read. Two attempts at scanning are necessary.

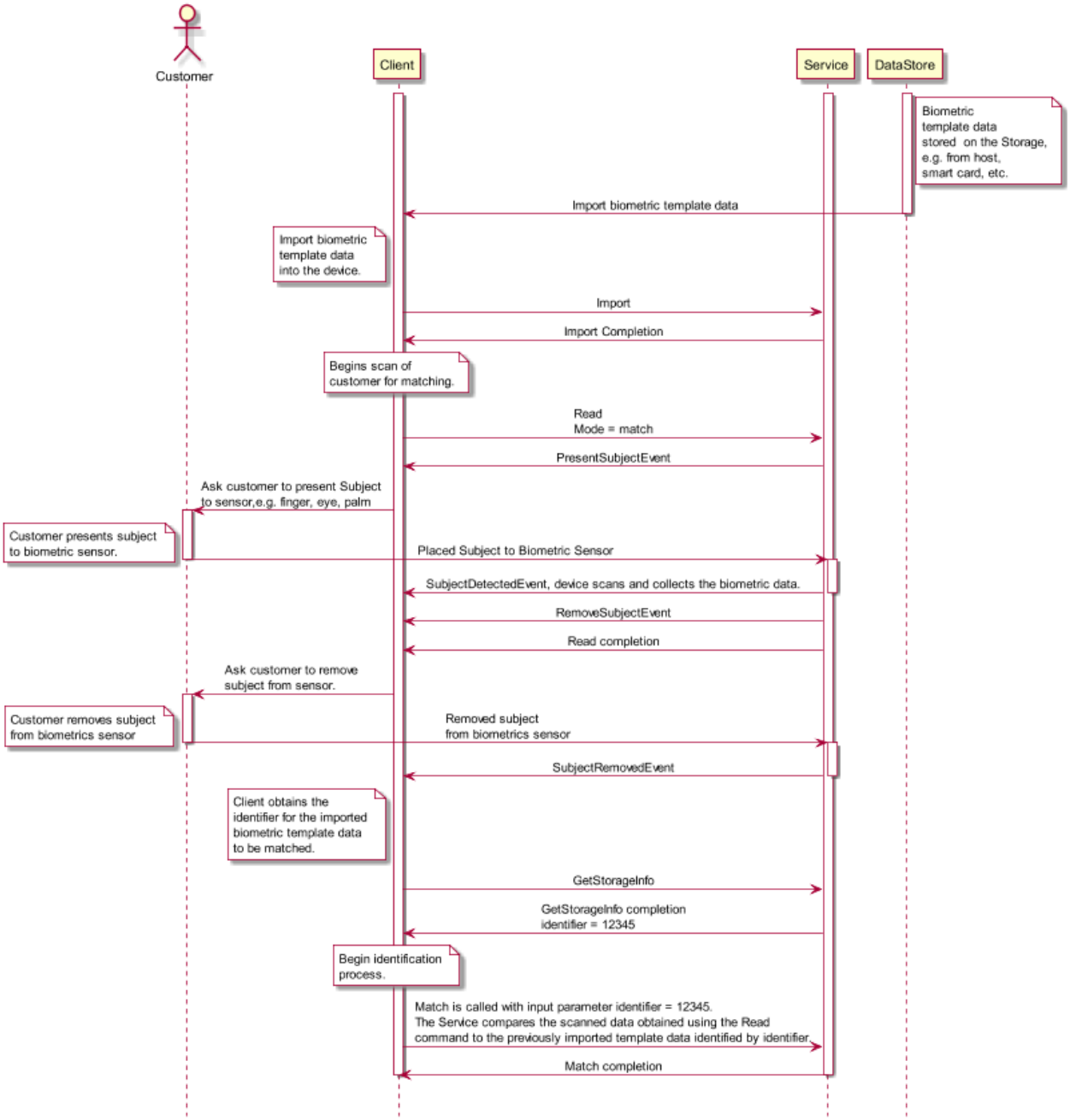


Biometric Match Command Flow – Separate Scan and Match

The following diagram 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, which assigns it a unique identifying number. This *identifier* number can then be retrieved using the Biometric.GetStorageInfo.

The Biometric.Read and Biometric.Match 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.Readis called to scan the subject's biometric data then the Biometric.Match is called to perform the match and return the result to the client.

In this case the capability matchSupported is reported as storedMatch.

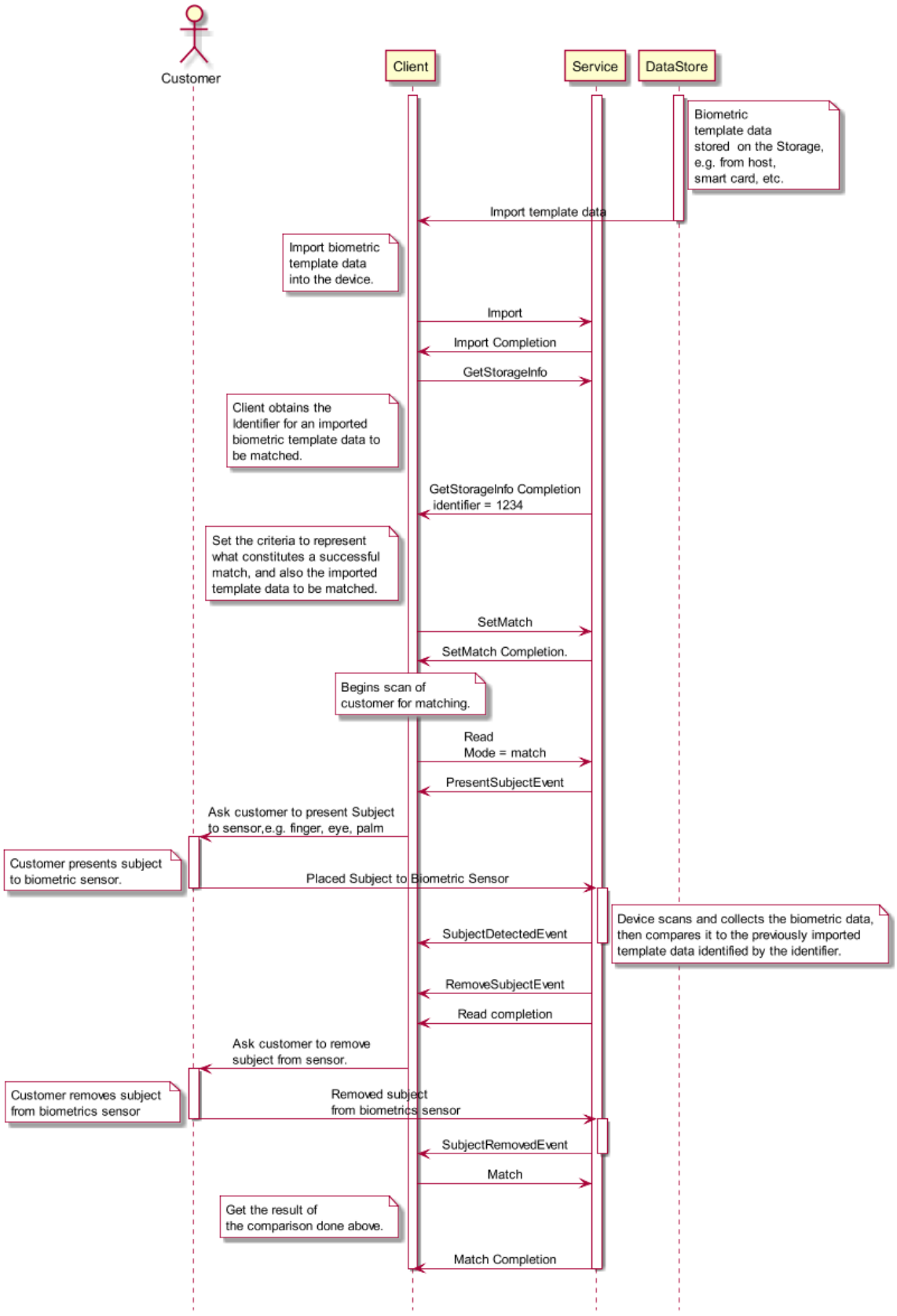


Biometric Match Command Flow – Combined Scan and Match

The following diagram 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, which assigns it a unique identifying number. This *identifier* number can then be retrieved using the Biometric.GetStorageInfo.

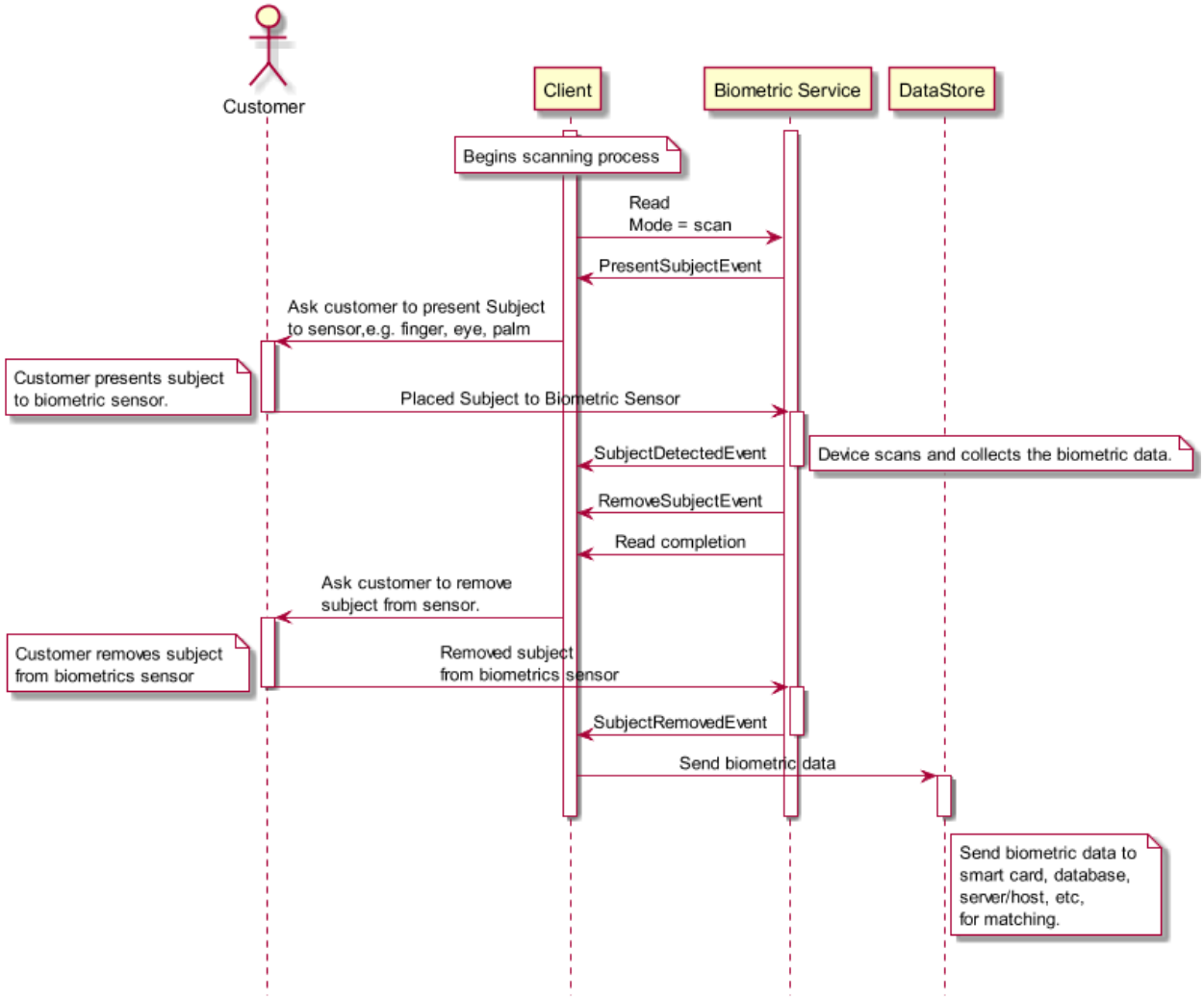
The Biometric.Read, Biometric.SetMatch and Biometric.Match are then used to scan data and compare it with the template identified by *identifier*. In this use case the device performs a combined scan and match operation, therefore the Biometric.SetMatch must be used to set the criteria to be used for matching, including the imported template to be identified by *identifier*. When the Biometric.Read is then called the device scans the user and performs the comparison as a combined operation. Finally the Biometric.Match is called to return the result of the comparison to the client.

In this case the capability matchSupported is reported as *combinedMatch*.



Biometric Scan-Only Command Flow

The following diagram 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 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.


Customer
Client
Biometric Service
DataStore
Begins scanning process
Read
Mode = scan
PresentSubjectEvent
Ask customer to present Subject
to sensor,e.g. finger, eye, palm
Customer presents subject
to biometric sensor.
Placed Subject to Biometric Sensor
SubjectDetectedEvent
Device scans and collects the biometric data.
RemoveSubjectEvent
Read completion
Ask customer to remove
subject from sensor.
Customer removes subject
from biometrics sensor
Removed subject
from biometrics sensor
SubjectRemovedEvent
Send biometric data
Send biometric data to
smart card, database,
server/host, etc,
for matching.

16.2 Command Messages

16.2.1 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 (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

timeout

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

default: 0

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "noImportedData",	string	
"templates": {	object	
"*id1*": {	object	
"format": "isoFid",	string	Yes
"algorithm": "ecb",	string	Yes
"keyName": "Key01"	string	
},		
"*id2*": {	object	
See id1 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:

- `noImportedData` - No data to return. Typically means that no data has been imported using the Biometric.Import.

Properties
templates A list of the biometric template data that were successfully imported. The object name of each biometric data type can be used in the *identifier* property for the Biometric.Match command.
templates/id1 (example name) Property name constraints: `pattern: ^id[0-9A-Za-z]+$`
templates/id1/format Specifies the format of the template data. Available values are described in the dataFormats. The following values are possible: • `isoFid` - Raw ISO FID format [Ref. biometric-3]. • `isoFmd` - ISO FMD template format [Ref. biometric-4]. • `ansiFid` - Raw ANSI FID format [Ref. biometric-1]. • `ansiFmd` - ANSI FMD template format [Ref. biometric-2]. • `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.
templates/id1/algorithm Specifies the encryption algorithm. This value is omitted if the biometric data is not encrypted. Available values are described in the encryptionalAlgorithm. 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.
templates/id1/keyName Specifies the name of the key that is used to encrypt the biometric data. This property is omitted if the biometric data is not encrypted. The detailed key information is available through the KeyManagement.GetKeyDetail.

Event Messages

None

16.2.2 Biometric.Read

This command enables the device for biometric scanning, then captures and optionally returns biometric data. A Biometric.PresentSubjectEvent will be sent to notify the client when it is ready to begin scanning and a Biometric.SubjectDetectedEventsent 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 in order to allow verification or identification of an individual. The matchSupportedcapability indicates if the Biometric.Match 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 property.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"dataTypes": [{	array (object)	
"format": "isoFid",	string	Yes
"algorithm": "ecb",	string	Yes
"keyName": "Key01"	string	
}],		
"numCaptures": 0,	integer	Yes
"mode": "scan"	string	Yes
}		

Properties

timeout

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

default: 0

dataTypes

Array of data types, 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 *dataTypes* 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.

Properties
dataTypes/format Specifies the format of the template data. Available values are described in the dataFormats. The following values are possible: • `isoFid` - Raw ISO FID format [Ref. biometric-3]. • `isoFmd` - ISO FMD template format [Ref. biometric-4]. • `ansiFid` - Raw ANSI FID format [Ref. biometric-1]. • `ansiFmd` - ANSI FMD template format [Ref. biometric-2]. • `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.
dataTypes/algorithm Specifies the encryption algorithm. This value is omitted if the biometric data is not encrypted. Available values are described in the encryptionalAlgorithm. 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.
dataTypes/keyName Specifies the name of the key that is used to encrypt the biometric data. This property is omitted if the biometric data is not encrypted. The detailed key information is available through the KeyManagement.GetKeyDetail.
numCaptures This property indicates the number of times to attempt capture of the biometric data from the subject. If this is zero or omitted, then the device determines how many attempts will be made. The maximum number of captures possible is indicated by the maxCapture capability. Property value constraints: `minimum: 0`
mode This optional property indicates the reason why the Biometric.Read has been issued, in order to allow for any necessary optimization. Available values are detailed in the scanModes. 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.

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	string	
`"errorCode": "readFailed",`	string	
`"dataRead": [{`	array (object)	
`"type": {`	object	Yes
`"format": "isoFid",`	string	Yes
`"algorithm": "ecb",`	string	Yes

Payload (version 1.0)	Type	Required
"keyName": "Key01"	string	
},		
"data": "1a987D000012Bb"	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.
- `modeNotSupported` - *mode* is not supported.
- `formatNotSupported` - The format specified is valid but not supported. A list of the supported values can be obtained through the dataFormats.
- `keyNotFound` - The specified key name is not found.

dataRead

This property is used to indicate the biometric data type of the template data contained. This property is not required if *dataTypes* property is omitted in the command payload.

dataRead/type

This property is used to indicate the biometric data type of the template data contained in *data*.

dataRead/type/format

Specifies the format of the template data. Available values are described in the dataFormats. The following values are possible:

- `isoFid` - Raw ISO FID format [Ref. biometric-3].
- `isoFmd` - ISO FMD template format [Ref. biometric-4].
- `ansiFid` - Raw ANSI FID format [Ref. biometric-1].
- `ansiFmd` - ANSI FMD template format [Ref. biometric-2].
- `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.

dataRead/type/algorithm

Specifies the encryption algorithm. This value is omitted if the biometric data is not encrypted. Available values are described in the encryptionalAlgorithm. 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.

dataRead/type/keyName

Specifies the name of the key that is used to encrypt the biometric data. This property is omitted if the biometric data is not encrypted. The detailed key information is available through the KeyManagement.GetKeyDetail.

Properties

dataRead/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

16.2.3 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. 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.Clearis 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 (version 1.0)	Type	Required
`{`		
`"timeout": 5000,`	integer	
`"templates": [{`	array (object)	
`"type": {`	object	Yes
`"format": "isoFid",`	string	Yes
`"algorithm": "ecb",`	string	Yes
`"keyName": "Key01"`	string	
`},`		
`"data": "1a987D000012Bb"`	string	Yes
`}]`		
`}`		

Properties

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

templates
Array of template data to be imported in the device.

templates/type
This property is used to indicate the biometric data type of the template data contained in *data*.

templates/type/format
Specifies the format of the template data. Available values are described in the dataFormats. The following values are possible:

- `isoFid` - Raw ISO FID format [Ref. biometric-3].
- `isoFmd` - ISO FMD template format [Ref. biometric-4].
- `ansiFid` - Raw ANSI FID format [Ref. biometric-1].
- `ansiFmd` - ANSI FMD template format [Ref. biometric-2].
- `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.

Properties
templates/type/algorithm Specifies the encryption algorithm. This value is omitted if the biometric data is not encrypted. Available values are described in the encryptionalAlgorithm. 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.
templates/type/keyName Specifies the name of the key that is used to encrypt the biometric data. This property is omitted if the biometric data is not encrypted. The detailed key information is available through the KeyManagement.GetKeyDetail.

templates/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 (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "invalidData",	string	
"templates": {	object	
"*id1*": {	object	
"format": "isoFid",	string	Yes
"algorithm": "ecb",	string	Yes
"keyName": "Key01"	string	
},		
"*id2*": {	object	
See id1 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.

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. • `formatNotSupported` - 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. • `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 remainingStorage. • `keyNotFound` - The specified key name is not found.
templates A list of the biometric template data that were successfully imported.
templates/id1 (example name) Property name constraints: `pattern: ^id[0-9A-Za-z]+$`
templates/id1/format Specifies the format of the template data. Available values are described in the dataFormats. The following values are possible: • `isoFid` - Raw ISO FID format [Ref. biometric-3]. • `isoFmd` - ISO FMD template format [Ref. biometric-4]. • `ansiFid` - Raw ANSI FID format [Ref. biometric-1]. • `ansiFmd` - ANSI FMD template format [Ref. biometric-2]. • `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.
templates/id1/algorithm Specifies the encryption algorithm. This value is omitted if the biometric data is not encrypted. Available values are described in the encryptionalAlgorithm. 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.
templates/id1/keyName Specifies the name of the key that is used to encrypt the biometric data. This property is omitted if the biometric data is not encrypted. The detailed key information is available through the KeyManagement.GetKeyDetail.

Event Messages

None

16.2.4 Biometric.Match

This command returns the result of a comparison between data that has been scanned using the Biometric.Readand template data that has been imported using the Biometric.Import. 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 and store it, then the data can be compared with imported biometric data using the Biometric.Match.

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 is compared with a single template that has been imported using the Biometric.Import. 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 is compared with multiple templates that have been imported using the Biometric.Import. 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"compareMode": "verify",	string	Yes
"identifier": "id1",	string	Yes
"maximum": 0,	integer	Yes
"threshold": 80	integer	Yes
}		

Properties

timeout
Timeout in milliseconds for the command to complete. If set to 0, 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.

Properties
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. If *compareMode* is identify a comparison is performed against all of the imported templates, in which case this property can be omitted. This property corresponds to the list of template identifiers returned by the Biometric.GetStorageInfo command. Property value constraints: `pattern: ^id[0-9A-Za-z]+$`
maximum Specifies the maximum number of matches to return. In the case where *compareMode* is verify this property can be omitted. Property value constraints: `minimum: 0`

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.
Property value constraints:
`minimum: 0`
`maximum: 100`

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "noImportedData",	string	
"candidates": {	object	
"*id1*": {	object	
"confidenceLevel": 0,	integer	
"templateData": "dGVtcGxhdGUgZGF0YQ=="	string	
},		
"*id2": {*	object	
See id1 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.

Properties
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. • `invalidIdentifier` - The command failed because data was imported but *identifier* was not found. • `modeNotSupported` - The type of match specified in *compareMode* is not supported. • `noCaptureData` - No captured data is present. Typically means that the Biometric.Readcommand has not been called, or the captured data has been cleared using the Biometric.Clear. • `invalidCompareMode` - The compare mode specified by the *compareMode* input parameter is not supported. • `invalidThreshold` - The *Threshold* input parameter is greater than the maximum allowed of 100.
candidates The object name has a unique number that positively identifies the biometric template data. This corresponds to the list of template identifiers returned by the Biometric.GetStorageInfo command. This property can be omitted if the Biometric.Match operation completes with no match found. If there are matches found, this property 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.
candidates/id1 (example name) Property name constraints: `pattern: ^id[0-9A-Za-z]+$`

candidates/id1/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* property.
Property value constraints:
`minimum: 0`
`maximum: 100`

candidates/id1/templateData
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.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

Event Messages

- Biometric.DataClearedEvent

16.2.5 Biometric.SetMatch

This command is used for devices which need to know the match criteria data for the Biometric.Matchbefore any biometric scanning is performed by the Biometric.Read. The 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 will fail with the generic error sequenceError. The data set using this command is not persistent across power failures.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"compareMode": "verify",	string	Yes
"identifier": "id1",	string	Yes
"maximum": 0,	integer	Yes
"threshold": 80	integer	Yes
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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. If *compareMode* is identify a comparison is performed against all of the imported templates, in which case this property can be omitted. This property corresponds to the list of template identifiers returned by the Biometric.GetStorageInfo command.

Property value constraints:

```
pattern: ^id[0-9A-Za-z]+$
```

maximum

Specifies the maximum number of matches to return. In the case where *compareMode* is verify this property can be omitted.

Property value constraints:

```
minimum: 0
```

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.

Property value constraints:

```
minimum: 0
maximum: 100
```

Completion Message

Payload (version 1.0)	Type	Required
{		

Payload (version 1.0)	Type	Required
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"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.
- `modeNotSupported` - The type of match specified in *compareMode* is not supported.
- `noImportedData` - The command failed because no data was imported previously using the Biometric.ImportData.
- `invalidThreshold` - The *threshold* input parameter is greater than the maximum allowed of 100.

Event Messages

None

16.2.6 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"clearData": "scannedData"	string	
}		

Properties

timeout

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

default: 0

clearData

This property indicates the type of data to be cleared from storage. If this property is omitted, then all stored data will be cleared. Available values are described in the clearData. The following values are possible:

- `scannedData` - Raw image data that has been scanned using the Biometric.Read can be cleared.
- `importedData` - Template data that was imported using the Biometric.Import can be cleared.
- `setMatchedData` - Match criteria data that was set using the Biometric.Match can be cleared.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

Properties

completionCode

The completion code.

errorDescription

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

Event Messages

- Biometric.DataClearedEvent

16.2.7 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"clearData": "scannedData"	string	
}		

Properties

timeout

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

default: 0

clearData

This property indicates the type of data to be cleared from storage. If this property is omitted, then all stored data will be cleared. Available values are described in the clearData. The following values are possible:

- `scannedData` - Raw image data that has been scanned using the Biometric.Read.
- `importedData` - Template data that was imported using the Biometric.Import.
- `setMatchedData` - Match criteria data that was set using the Biometric.Match.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

Properties

completionCode

The completion code.

errorDescription

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

Event Messages

- Biometric.DataClearedEvent

16.2.8 Biometric.SetDataPersistence

This command is used to set the persistence mode. This controls how the biometric data is persisted after a Biometric.Read. The data can be persisted for use by subsequent commands, or it can be automatically cleared.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"persistenceMode": "persist"	string	Yes
}		

Properties

timeout

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

default: 0

persistenceMode

Specifies the data persistence mode. This controls how biometric data that has been captured using the Biometric.Readcommand will persist. This value itself is persistent. Available values are described in the persistenceModes. The following values are possible:

- `persist` - Biometric data captured using the Biometric.Read can persist until all sessions are closed, the device is power failed or rebooted, or the Biometric.Readis requested again. This captured biometric data can also be explicitly cleared using the Biometric.Clear or Biometric.Reset.
- `clear` - Captured biometric data will not persist. Once the data has been either returned in the Biometric.Readcommand or used by the Biometric.Match, then the data is cleared from the device.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	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.

errorCode

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

- `modeNotSupported` - The command failed because a mode was specified which is not supported.

Event Messages

None

16.3 Unsolicited Messages

16.3.1 Biometric.PresentSubjectEvent

This 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.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
}		

16.3.2 Biometric.SubjectDetectedEvent

This 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.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
}		

16.3.3 Biometric.RemoveSubjectEvent

This event is used to notify a client that the subject should be removed from the capture area of the device.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
}		

16.3.4 Biometric.SubjectRemovedEvent

This message is generated when the subject has been removed from the capture area of the device. This event may be generated at any time.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
}		

16.3.5 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 completion, or as a result of an explicit call to the Biometric.Clear or Biometric.Reset.

Unsolicited Message

Payload (version 1.0)	Type	Required
`{`		
`"clearData": "scannedData"`	string	
`}`		

Properties

clearData

This parameter indicates the type of data that has been cleared from storage. If this property is omitted, then all stored data has been cleared. Available values are described in the clearData. The following values are possible:

- `scannedData` - Raw image data that has been scanned using the Biometric.Read.
- `importedData` - Template data that was imported using the Biometric.Import.
- `setMatchedData` - Match criteria data that was set using the Biometric.Match.

16.3.6 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.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
}		

17. Camera Interface

This chapter defines the Camera interface functionality and messages under XFS4IoT.

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 as a set of cameras, one for each of its possible positions.

17.1 Command Messages

17.1.1 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* property.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"camera": "room",	string	
"camData": "Camera 1 Text",	string	
"pictureFile": "Xhdjyedh736ydw7hdi"	string	
}		

Properties

timeout
Timeout in milliseconds for the command to complete. If set to 0, 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 supported by manAdd. 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.

pictureFile
The base64 encoded data representing the picture. This is only supported if pictureFileis true.
Property value constraints:

```
pattern: ^[A-Za-z0-9+/]+={0,2}$
format: base64
```

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

Properties

completionCode
The completion code.

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

Event Messages

- Camera.InvalidDataEvent

17.1.2 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 (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

Properties

completionCode
The completion code.

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

Event Messages

None

17.2 Event Messages

17.2.1 Camera.InvalidDataEvent

This event is used to specify that the text string given was too long or in some other way invalid.

Event Message

Payload (version 1.0)	Type	Required
{		
}		

17.3 Unsolicited Messages

17.3.1 Camera.MediaThresholdEvent

This event is used to specify that the state of the recording media reached a threshold.

Unsolicited Message

Payload (version 1.0)	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.

18. Lights Interface

This chapter defines the Lights interface functionality and messages.

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.

18.1 Command Messages

18.1.1 Lights.SetLight

This command is used to set the status of a light.

For guidance lights, 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"cardReader": {	object	
"position": "left",	string	
"flashRate": "off",	string	
"color": "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.		
},		
"billAcceptor": {	object	
See cardReader properties.		
},		
"envelopeDispenser": {	object	
See cardReader properties.		
},		
"documentPrinter": {	object	

Payload (version 1.0)	Type	Required
See cardReader properties.		
},		
"coinAcceptor": {	object	
See cardReader properties.		
},		
"scanner": {	object	
See cardReader properties.		
},		
"contactless": {	object	
See cardReader properties.		
},		
"cardReader2": {	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.		
},		
"fasciaLight": {	object	
See cardReader properties.		
},		
"*vendorSpecificLight*": {	object	
See cardReader properties.		
}		

Payload (version 1.0)	Type	Required
}		

Properties

timeout

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

default: 0

cardReader

Card Reader Light.

cardReader/position

The light position. Can be used for devices which have multiple input and output positions, omitted if not required. 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).

cardReader/color

The light color as one of the following values:

- `red` - The light is red.
- `green` - The light is green.
- `yellow` - The light is 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.

Properties
envelopeDepository Envelope Depository Light.
billAcceptor Bill Acceptor Light.
envelopeDispenser Envelope Dispenser Light.
documentPrinter Document Printer Light.
coinAcceptor Coin Acceptor Light.
scanner Scanner Light.
contactless Contactless Reader Light.
cardReader2 Card Reader 2 Light.
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.
vendorSpecificLight (example name) Additional vendor specific lights

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "invalidLight"	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: • `invalidLight` - An attempt to set a light to a new value was invalid because the light does not exist. • `lightError` - A hardware error occurred while executing the command.

Event Messages

None

19. Auxiliaries Interface

This chapter defines the Auxiliaries interface functionality and messages.

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.

19.1 Command Messages

19.1.1 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 (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	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.

Properties
mode Specifies the current or desired 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 properties 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 or desired auto start-up time configuration.

startTime/year
Specifies the year if relevant to the *mode*.
Property value constraints:
`minimum: 1601`
`maximum: 30827`

startTime/month
Specifies the month if relevant to the *mode*.
Property value constraints:
`minimum: 1`
`maximum: 12`

startTime/dayOfWeek Specifies the day of the week, if relevant to the *mode*. 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 if relevant to the *mode*.
Property value constraints:
`minimum: 1`
`maximum: 31`

startTime/hour
Specifies the hour if relevant to the *mode*.
Property value constraints:
`minimum: 0`
`maximum: 23`

startTime/minute
Specifies the minute if relevant to the *mode*.
Property value constraints:
`minimum: 0`
`maximum: 59`

Event Messages

None

19.1.2 Auxiliaries.ClearAutoStartupTime

This command is used to clear the time at which the machine will automatically start.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

Properties

completionCode
The completion code.

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

Event Messages

None

19.1.3 Auxiliaries.Register

This command is used to register or deregister for 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. Only the events specified in the input payload will be affected, all others will remain in the same state.

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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"operatorSwitch": "register",	string	
"tamperSensor": "register",	string	
"internalTamperSensor": "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	
"volumeControl": "register",	string	
"ups": "register",	string	
"remoteStatusMonitor": "register",	string	
"audibleAlarm": "register",	string	

Payload (version 1.0)	Type	Required
"enhancedAudioControl": "register",	string	
"enhancedMicrophoneControl": "register",	string	
"microphoneVolume": "register",	string	
"*exampleProperty1*": "register",	string	
"*exampleProperty2*": "register"	string	
}		

Properties

timeout

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

default: 0

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.

internalTamperSensor

Specifies whether the Internal Tamper Sensor should report whenever someone tampers with the internal alarm. See operatorSwitch for the possible values.

seismicSensor

Specifies whether the Seismic Sensor should report whenever any seismic activity is detected. See operatorSwitch for the possible values.

heatSensor

Specifies whether the Heat Sensor should report whenever any excessive heat is detected. See operatorSwitch for the possible values.

proximitySensor

Specifies whether the Proximity Sensor should report whenever any movement is detected close to the terminal. See operatorSwitch for the possible values.

ambientLightSensor

Specifies whether the Ambient Light Sensor should report whenever it detects changes in the ambient light. See operatorSwitch for the possible values.

enhancedAudio

Specifies whether the Audio Jack should report whenever it detects changes in the audio jack. See operatorSwitch for the possible values.

bootSwitch

Specifies whether the Boot Switch should report whenever the delayed effect boot switch is used. See operatorSwitch for the possible values.

consumerDisplay

Specifies whether the Consumer Display Sensor should report whenever it detects changes to the consumer display. See operatorSwitch for the possible values.

operatorCallButton

Specifies whether the Operator Call Button should report whenever the Operator Call Button is pressed or released. See operatorSwitch for the possible values.

Properties
handsetSensor Specifies whether the Handset Sensor should report whenever it detects changes of its status. See operatorSwitch for the possible values.
headsetMicrophone Specifies whether the Microphone Jack should report whenever it detects changes in the microphone jack. See operatorSwitch for the possible values.
cabinetDoor Specifies whether the Cabinet Doors should report whenever the doors are opened, closed, bolted or locked. See operatorSwitch for the possible values.
safeDoor Specifies whether the Safe Doors should report whenever the doors are opened, closed, bolted or locked. See operatorSwitch for the possible values.
vandalShield Specifies whether the Vandal Shield should report whenever the shield changed position. See operatorSwitch for the possible values.
cabinetFront Specifies whether the front Cabinet Doors should report whenever the front doors are opened, closed, bolted or locked. See operatorSwitch for the possible values.
cabinetRear Specifies whether the rear Cabinet Doors should report whenever the front doors are opened, closed, bolted or locked. See operatorSwitch for the possible values.
cabinetRight Specifies whether the right Cabinet Doors should report whenever the front doors are opened, closed, bolted or locked. See operatorSwitch for the possible values.
cabinetLeft Specifies whether the left Cabinet Doors should report whenever the front doors are opened, closed, bolted or locked. See operatorSwitch for the possible values.
openCloseIndicator Specifies whether the Open/Closed Indicator should report whenever it is turned on (set to open) or turned off (set to closed). See operatorSwitch for the possible values.
fasciaLight Specifies whether the Fascia Light should report whenever it is turned on or turned off. See operatorSwitch for the possible values.
audioIndicator Specifies whether the Audio Indicator should report whenever it is turned on or turned off. See operatorSwitch for the possible values.
heatingIndicator Specifies whether the Heating device should report whenever it is turned on or turned off. See operatorSwitch for the possible values.
consumerDisplayBacklight Specifies whether the Consumer Display Backlight should report whenever it is turned on or turned off. See operatorSwitch for the possible values.
signageDisplay Specifies whether the Signage Display should report whenever it is turned on or turned off. See operatorSwitch for the possible values.
volumeControl Specifies whether the Volume Control device should report whenever it is changed. See operatorSwitch for the possible values.

Properties
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 (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	string	
`"errorCode": "invalidAuxiliary"`	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: • `invalidAuxiliary` - An attempt to register for or disable events to a auxiliary was invalid because the auxiliary does not exist.

Event Messages

None

19.1.4 Auxiliaries.SetAuxiliaries

This command is used to set or clear one or more device auxiliaries.

Command Message

Payload (version 1.0)	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 0, 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.

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.

Properties
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 all the 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.
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.

Properties
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.
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.

Properties
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`

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"errorCode": "invalidAuxiliary"	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: • `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.

Event Messages

- Auxiliaries.AuxiliaryStatusEvent

19.1.5 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 (version 1.0)	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 0, the command will not timeout but can be canceled.

default: 0

mode

Specifies the current or desired 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 properties 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 or desired auto start-up time configuration.

startTime/year

Specifies the year if relevant to the *mode*.

Property value constraints:

```
minimum: 1601
maximum: 30827
```

startTime/month

Specifies the month if relevant to the *mode*.

Property value constraints:

```
minimum: 1
maximum: 12
```

Properties
startTime/dayOfWeek Specifies the day of the week, if relevant to the *mode*. 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 if relevant to the *mode*.
Property value constraints:
`minimum: 1`
`maximum: 31`

startTime/hour
Specifies the hour if relevant to the *mode*.
Property value constraints:
`minimum: 0`
`maximum: 23`

startTime/minute
Specifies the minute if relevant to the *mode*.
Property value constraints:
`minimum: 0`
`maximum: 59`

Completion Message

Payload (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available"`	string	
`}`		

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

Event Messages

None

19.2 Unsolicited Messages

19.2.1 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.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
"operatorSwitch": "notAvailable",	string	
"tamperSensor": "notAvailable",	string	
"internalTamperSensor": "notAvailable",	string	
"seismicSensorState": "notAvailable",	string	
"heatSensorState": "notAvailable",	string	
"proximitySensorState": "notAvailable",	string	
"ambientLightSensorState": "notAvailable",	string	
"enhancedAudioSensorState": "notAvailable",	string	
"bootSwitchSensorState": "notAvailable",	string	
"displaySensorState": "notAvailable",	string	
"operatorCallButtonSensorState": "notAvailable",	string	
"handsetSensorState": "notAvailable",	string	
"headsetMicrophoneSensorState": "notAvailable",	string	
"fasciaMicrophoneSensorState": "notAvailable",	string	
"safeDoorState": "notAvailable",	string	
"vandalShieldState": "notAvailable",	string	
"cabinetFrontDoorState": "notAvailable",	string	
"cabinetRearDoorState": "notAvailable",	string	
"cabinetLeftDoorState": "notAvailable",	string	
"cabinetRightDoorState": "notAvailable",	string	
"openClosedIndicatorState": "notAvailable",	string	
"audioState": {	object	
"rate": "on",	string	
"signal": "keypress"	string	
},		
"heatingState": "notAvailable",	string	
"consumerDisplayBacklightState": "notAvailable",	string	
"signageDisplayState": "notAvailable",	string	
"volumeState": {	object	
"volumeLevel": 1	integer	
},		
"upsState": {	object	

Payload (version 1.0)	Type	Required
"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` - The switch is in supervisor mode.

tamperSensor

Specifies the state of the Tamper sensor.

- `notAvailable` - The status is not available.
- `off` - There is no indication of a tampering attempt.
- `on` - There has been a tampering attempt.

internalTamperSensor

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 status 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 is not available.
- `off` - The 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.

Properties
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.
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.

Properties
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.
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 fully 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. • `partiallyOpen` - The Vandal Shield is partially open. • `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.

Properties
cabinetRearDoorState Specifies the overall state of the Rear Cabinet Doors. The rear 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.
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.
audioState/rate Specifies the state 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.
audioState/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.

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. 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 Specifies the state of the volume control. Omitted if not available.

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. Omitted if the status is not available. • `low` - The charge level of the UPS is low. • `engaged` - The UPS is engaged. • `powering` - The UPS is powering the system. • `recovered` - The UPS was engaged when the main power went off.
upsState/low default: false
upsState/engaged default: false
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.

Properties

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` - The 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.

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.

Properties
microphoneVolumeState/available Specifies whether the Microphone 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`

20. Storage Interface

This chapter defines the Storage interface functionality and messages.

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 Cash Dispenser, Cash Acceptor or Card Reader interfaces to handle management of the device storage units.

20.1 General Information

20.1.1 Transaction Flows

The following sections describe how various scenarios are handled using XFS4IoT Storage.

Replenishment of a Cash Handling device

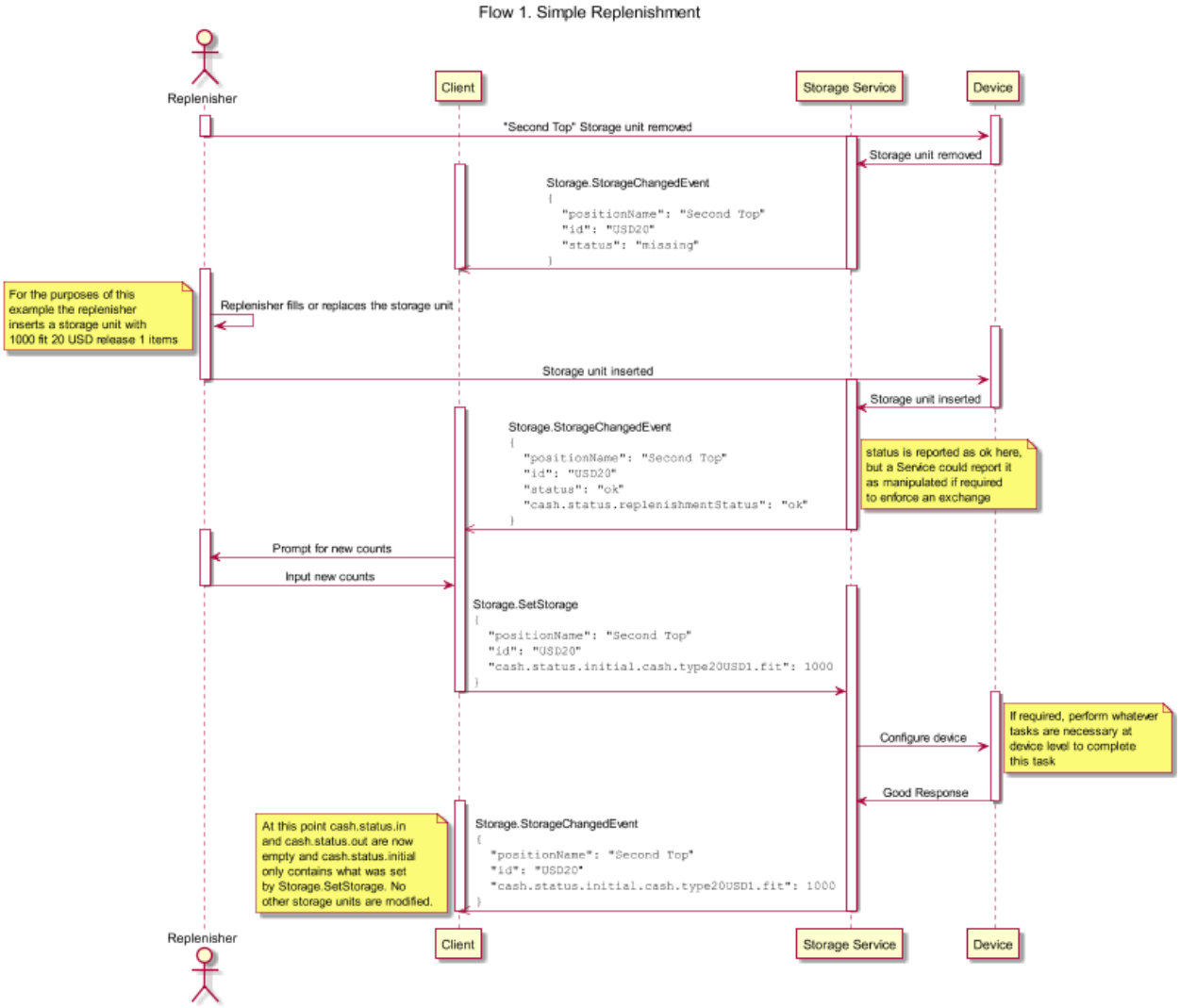
Manual Cash Replenishment in XFS4IoT is performed using the Storage.SetStorage command.

Storage.SetStorage can operate in two flows depending on whether the associated 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.

20.1.1.1.1 Flow 1 - No Exchange

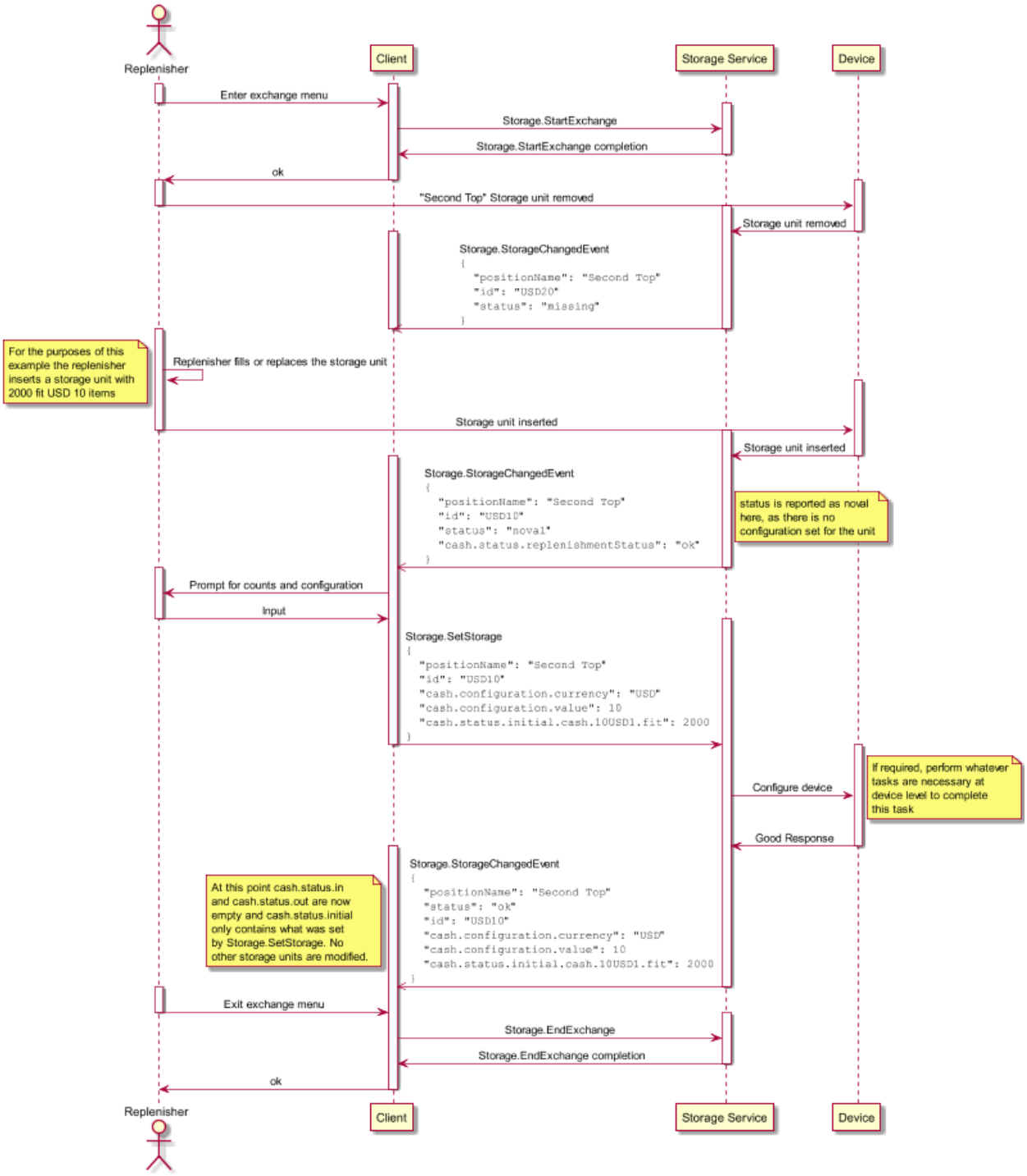
In flow 1, one or more storage units are replenished and as only counts have changed, an exchange session is not required. In this scenario the replenisher removes a storage unit and replaces it with one which contains 1000 USD 20 notes.



20.1.1.1.2 Flow 2 - With Exchange

In flow 2, a storage unit needs to be configured, therefore an exchange session is required. In this scenario the replenisher removes the storage unit used in flow 1 and replaces it with a different one which contains 1000 USD 20 notes.

Flow 2. Replenishment Including Exchange



20.2 Command Messages

20.2.1 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.

This command can be used while in Vendor Mode.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"storage": {	object	
"*unit1*": {	object	
"id": "RC1",	string	
"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	

Payload (version 1.0)	Type	Required
"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": ["type20USD1", "type50USD1"]	array (string)	
},		
"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": ["type20USD1", "type50USD1"],	array (string)	
"name": "$10",	string	
"maxRetracts": 5	integer	
},		
"status": {	object	
"index": 4,	integer	
"initial": {	object	
"unrecognized": 0,	integer	
"*type20USD1*": {	object	
"fit": 0,	integer	
"unfit": 0,	integer	
"suspect": 0,	integer	
"counterfeit": 0,	integer	
"inked": 0	integer	
},		

Payload (version 1.0)	Type	Required
"*type50USD1": {*	object	
See type20USD1 properties.		
}		
},		
"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	
See initial properties.		
},		

Payload (version 1.0)	Type	Required
"transport": {	object	
See initial properties.		
}		
},		
"accuracy": "notSupported",	string	
"replenishmentStatus": "ok",	string	
"operationStatus": "dispenseInoperative"	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

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: ^unit[0-9A-Za-z]+$
```

Properties
storage/unit1/id An identifier which can be used for cUnitID in CDM/CIM XFS 3.x migration. Not required if not applicable. Property value 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.
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.

Properties
storage/unit1/cash/capabilities/items/fit The storage unit can store cash items which are fit for recycling.
storage/unit1/cash/capabilities/items/unfit The storage unit can store cash items which are unfit for recycling.
storage/unit1/cash/capabilities/items/unrecognized The storage unit can store unrecognized cash items.
storage/unit1/cash/capabilities/items/counterfeit The storage unit can store counterfeit cash items.
storage/unit1/cash/capabilities/items/suspect The storage unit can store suspect counterfeit cash items.
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. If true, then hardware sensors will override count-based replenishment status for *empty* and *full*. Other replenishment states can be overridden by counts.
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 An array containing multiple cash items, listing what a storage unit is physically capable of handling. For example a coin storage unit may be restricted to one coin denomination due to the hardware. Each member is the object name of a cash item reported by CashManagement.GetBankNoteTypes.
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.

Properties
storage/unit1/cash/configuration/currency ISO 4217 currency identifier [Ref. cashmanagement-1]. May only be modified in an exchange state if applicable. May be omitted if the unit is configured to store mixed currencies or non-cash items. Property value constraints: `pattern: ^[A-Z]{3}$`
storage/unit1/cash/configuration/value Absolute value of a cash item or items. 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. If applied to a storage unit, this applies to all contents, may be 0 if mixed and may only be modified in an exchange state if applicable. Property value constraints: `minimum: 0`
storage/unit1/cash/configuration/highThreshold If specified, replenishmentStatusis set to *high* if the total number of items in the storage unit is greater than this number. The total number is not reported directly, but derived from *initial + in - out*. 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, replenishmentStatusis set to *low* if total number of items in the storage unit is less than this number. The total number is not reported directly, but derived from *initial + in - out*. 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 array containing multiple cash items, listing what a storage unit is capable of or configured to handle. Each member is the object name of a cash item reported by CashManagement.GetBankNoteTypes.
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. Only applies to retract units. If retractOperationsequals this number, then no further retracts are allowed into this storage 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. Note that the count of items in the storage unit must be derived from the counts reported.
storage/unit1/cash/status/index Assigned by the Service. Will be a unique number which can be used to determine *usNumber* in XFS 3.x migration. This can change as storage units are added and removed from the storage collection. Property value constraints: `minimum: 1`

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/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification.
storage/unit1/cash/status/initial/type20USD1/fit Count of genuine cash items which are fit for recycling.
storage/unit1/cash/status/initial/type20USD1/unfit Count of genuine cash items which are unfit for recycling.
storage/unit1/cash/status/initial/type20USD1/suspect Count of suspected counterfeit cash items.
storage/unit1/cash/status/initial/type20USD1/counterfeit Count of counterfeit cash items.
storage/unit1/cash/status/initial/type20USD1/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. This includes intermediate positions such as a stacker, where an item has been moved before moving to the final destination such as another storage unit or presentation to a customer. Counts for non-intermediate positions are reset if initial is set for this unit by Storage.GetStorage. See descriptions for the counts which will not be reset by this command. Intermediate position counts are reset when the intermediate position is empty: • If it is known where the items moved to then the appropriate count or counts are modified. • If it is not known where the items moved, for example because they have been removed manually after jam clearance, then *unknown* is modified.
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. This is not reset if initial is set for this unit by Storage.GetStorage.
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. This is not reset if initial is set for this unit by Storage.GetStorage.

Properties
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. This is not reset if initial is set for this unit by Storage.GetStorage.
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. This also reports items in the *transport*, where an item has jammed before being deposited in the storage unit. Counts other than *transport* are reset if initial is set for this unit by Storage.GetStorage. See descriptions for the counts which will not be reset by this command. The *transport* count is reset when it is empty: • If it is known where the items moved to then the appropriate count or counts are modified. • If it is not known where the items moved, for example because they have been removed manually after jam clearance, then *unknown* is modified.
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 retracted into 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/in/transport The items which were intended to be deposited in this storage unit but are not yet deposited. Typical use case for this property is tracking items after a jam during CashAcceptor.CashInEnd. This is not reset if initial is set for this unit by Storage.GetStorage.
storage/unit1/cash/status/accuracy Describes the accuracy of the counts reported by *out* and *in*. 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.

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. This is based on hardware detection, either on sensors or counts. • `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. If the storage unit has hardwareSensors, this state is not set by counts.
storage/unit1/cash/status/operationStatus On some devices it may be possible to allow items to be dispensed in a recycling storage unit while deposit is inoperable or vice-versa. This property allows the Service to report that one operation is possible while the other is not, without taking the storage unit out of Service completely with status or replenishmentStatus. Following values are possible: • `dispenseInoperative` - Dispense operations are possible and deposit operations are not possible on this recycling storage unit. • `depositInoperative` - Deposit operations are possible and dispense operations are not possible on this recycling storage unit. If not reported, *status* and *replenishmentStatus* apply to both cash out and cash in 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 Indicates whether 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.

Properties
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`
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

20.2.2 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 (version 1.0)	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	
"value": 20,	number	
"highThreshold": 500,	integer	
"lowThreshold": 10,	integer	
"appLockIn": false,	boolean	

Payload (version 1.0)	Type	Required
"appLockOut": false,	boolean	
"cashItems": ["type20USD1", "type50USD1"],	array (string)	
"name": "$10",	string	
"maxRetracts": 5	integer	
},		
"status": {	object	
"initial": {	object	
"unrecognized": 0,	integer	
"*type20USD1*": {	object	
"fit": 0,	integer	
"unfit": 0,	integer	
"suspect": 0,	integer	
"counterfeit": 0,	integer	
"inked": 0	integer	
},		
"*type50USD1": {*	object	
See type20USD1 properties.		
}		
}		
}		
},		
"card": {	object	
"configuration": {	object	
"cardID": "LoyaltyCard",	string	
"threshold": 0	integer	
},		
"status": {	object	
"initialCount": 0	integer	
}		
}		
},		
"*unit2": {*	object	
See unit1 properties.		
}		
}		
}		

Properties

timeout

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

default: 0

Properties
storage Object containing storage unit information.
storage/unit1 (example name) The object contains a single storage unit. Property name constraints: `pattern: ^unit[0-9A-Za-z]+$`
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.
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.
storage/unit1/cash/configuration/items/unfit The storage unit can store cash items which are unfit for recycling.
storage/unit1/cash/configuration/items/unrecognized The storage unit can store unrecognized cash items.
storage/unit1/cash/configuration/items/counterfeit The storage unit can store counterfeit cash items.
storage/unit1/cash/configuration/items/suspect The storage unit can store suspect counterfeit cash items.
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.

Properties
storage/unit1/cash/configuration/items/document Storage unit containing documents.
storage/unit1/cash/configuration/currency ISO 4217 currency identifier [Ref. cashmanagement-1]. May only be modified in an exchange state if applicable. May be omitted if the unit is configured to store mixed currencies or non-cash items. Property value constraints: `pattern: ^[A-Z]{3}$`
storage/unit1/cash/configuration/value Absolute value of a cash item or items. 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. If applied to a storage unit, this applies to all contents, may be 0 if mixed and may only be modified in an exchange state if applicable. Property value constraints: `minimum: 0`
storage/unit1/cash/configuration/highThreshold If specified, replenishmentStatusis set to *high* if the total number of items in the storage unit is greater than this number. The total number is not reported directly, but derived from *initial + in - out*. 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, replenishmentStatusis set to *low* if total number of items in the storage unit is less than this number. The total number is not reported directly, but derived from *initial + in - out*. 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 array containing multiple cash items, listing what a storage unit is capable of or configured to handle. Each member is the object name of a cash item reported by CashManagement.GetBankNoteTypes.
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. Only applies to retract units. If retractOperationsequals this number, then no further retracts are allowed into this storage 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 inare reset to empty.
storage/unit1/cash/status/initial/unrecognized Count of unrecognized items handled by the cash interface.
storage/unit1/cash/status/initial/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification.

Properties
storage/unit1/cash/status/initial/type20USD1/fit Count of genuine cash items which are fit for recycling.
storage/unit1/cash/status/initial/type20USD1/unfit Count of genuine cash items which are unfit for recycling.
storage/unit1/cash/status/initial/type20USD1/suspect Count of suspected counterfeit cash items.
storage/unit1/cash/status/initial/type20USD1/counterfeit Count of counterfeit cash items.
storage/unit1/cash/status/initial/type20USD1/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 (version 1.0)	Type	Required
`{`		
`"completionCode": "success",`	string	
`"errorDescription": "Device not available",`	string	
`"errorCode": "invalidUnit"`	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.

Properties
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

20.2.3 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.StatusChangedEvent.

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 storage unit information, until the *Storage.EndExchange* command is performed.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	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.

Properties
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

20.2.4 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 storage 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.StatusChangedEvent.

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.StorageErrorEvent will be sent for any storage unit which cannot be successfully updated. If no units could be updated then an error code will be returned.

Even if this command does not return a successful completion the exchange state has ended.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	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.

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

20.3 Unsolicited Messages

20.3.1 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.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
"*unit1*": {	object	
"id": "RC1",	string	
"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	

Payload (version 1.0)	Type	Required
"cashItems": ["type20USD1", "type50USD1"]	array (string)	
},		
"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": ["type20USD1", "type50USD1"],	array (string)	
"name": "$10",	string	
"maxRetracts": 5	integer	
},		
"status": {	object	
"index": 4,	integer	
"initial": {	object	
"unrecognized": 0,	integer	
"*type20USD1*": {	object	
"fit": 0,	integer	
"unfit": 0,	integer	
"suspect": 0,	integer	
"counterfeit": 0,	integer	
"inked": 0	integer	
},		
"*type50USD1": {*	object	
See type20USD1 properties.		
}		
},		
"out": {	object	
"presented": {	object	
See initial properties.		
},		
"rejected": {	object	
See initial properties.		
},		

Payload (version 1.0)	Type	Required
"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	
See initial properties.		
},		
"transport": {	object	
See initial properties.		
}		
},		
"accuracy": "notSupported",	string	
"replenishmentStatus": "ok",	string	
"operationStatus": "dispenseInoperative"	string	
}		
},		
"card": {	object	
"capabilities": {	object	

Payload (version 1.0)	Type	Required
"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	
}		
}		
}		
}		

Properties

unit1 (example name)

The object contains a single storage unit.

Property name constraints:

```
pattern: ^unit[0-9A-Za-z]+$
```

unit1/id

An identifier which can be used for cUnitID in CDM/CIM XFS 3.x migration. Not required if not applicable.

Property value constraints:

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

unit1/positionName

Fixed physical name for the position.

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
```

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.

unit1/serialNumber

The storage unit's serial number if it can be read electronically.

unit1/cash

The cash related contents, status and configuration of the unit.

Properties
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.
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.
unit1/cash/capabilities/types/cashIn The unit can accept cash items. If *cashOut* is also true then the unit can recycle.
unit1/cash/capabilities/types/cashOut The unit can dispense cash items. If *cashIn* is also true then the unit can recycle.
unit1/cash/capabilities/types/replenishment Replenishment container. A storage unit can be refilled from or emptied to a replenishment container.
unit1/cash/capabilities/types/cashInRetract Retract unit. Items can be retracted into this unit during Cash In operations.
unit1/cash/capabilities/types/cashOutRetract Retract unit. Items can be retracted into this unit during Cash Out operations.
unit1/cash/capabilities/types/reject Reject unit. Items can be rejected into this unit.
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.
unit1/cash/capabilities/items/fit The storage unit can store cash items which are fit for recycling.
unit1/cash/capabilities/items/unfit The storage unit can store cash items which are unfit for recycling.
unit1/cash/capabilities/items/unrecognized The storage unit can store unrecognized cash items.
unit1/cash/capabilities/items/counterfeit The storage unit can store counterfeit cash items.
unit1/cash/capabilities/items/suspect The storage unit can store suspect counterfeit cash items.
unit1/cash/capabilities/items/inked The storage unit can store cash items which have been identified as ink stained.
unit1/cash/capabilities/items/coupon Storage unit containing coupons or advertising material.
unit1/cash/capabilities/items/document Storage unit containing documents.
unit1/cash/capabilities/hardwareSensors Indicates whether the storage unit has sensors which report the status. If true, then hardware sensors will override count-based replenishment status for *empty* and *full*. Other replenishment states can be overridden by counts.

Properties
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`
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.
unit1/cash/capabilities/cashItems An array containing multiple cash items, listing what a storage unit is physically capable of handling. For example a coin storage unit may be restricted to one coin denomination due to the hardware. Each member is the object name of a cash item reported by CashManagement.GetBankNoteTypes.
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.
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.
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.
unit1/cash/configuration/currency ISO 4217 currency identifier [Ref. cashmanagement-1]. May only be modified in an exchange state if applicable. May be omitted if the unit is configured to store mixed currencies or non-cash items. Property value constraints: `pattern: ^[A-Z]{3}$`
unit1/cash/configuration/value Absolute value of a cash item or items. 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. If applied to a storage unit, this applies to all contents, may be 0 if mixed and may only be modified in an exchange state if applicable. Property value constraints: `minimum: 0`
unit1/cash/configuration/highThreshold If specified, replenishmentStatusis set to *high* if the total number of items in the storage unit is greater than this number. The total number is not reported directly, but derived from *initial + in - out*. If not specified, high is based on hardware sensors if supported - see hardwareSensors. Property value constraints: `minimum: 1`
unit1/cash/configuration/lowThreshold If specified, replenishmentStatusis set to *low* if total number of items in the storage unit is less than this number. The total number is not reported directly, but derived from *initial + in - out*. If not specified, low is based on hardware sensors if supported - see hardwareSensors. Property value constraints: `minimum: 1`

Properties
unit1/cash/configuration/appLockIn If true, items cannot be accepted into the storage unit in Cash In operations.
unit1/cash/configuration/appLockOut If true, items cannot be dispensed from the storage unit in Cash Out operations.
unit1/cash/configuration/cashItems An array containing multiple cash items, listing what a storage unit is capable of or configured to handle. Each member is the object name of a cash item reported by CashManagement.GetBankNoteTypes.
unit1/cash/configuration/name Application configured name of the unit.
unit1/cash/configuration/maxRetracts If specified, this is the number of retract operations allowed into the unit. Only applies to retract units. If retractOperationsequals this number, then no further retracts are allowed into this storage unit. If not specified, the maximum number is not limited by counts. Property value constraints: `minimum: 1`
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. Note that the count of items in the storage unit must be derived from the counts reported.
unit1/cash/status/index Assigned by the Service. Will be a unique number which can be used to determine *usNumber* in XFS 3.x migration. This can change as storage units are added and removed from the storage collection. Property value constraints: `minimum: 1`
unit1/cash/status/initial The cash related items which were in the storage unit at the last replenishment.
unit1/cash/status/initial/unrecognized Count of unrecognized items handled by the cash interface.
unit1/cash/status/initial/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification.
unit1/cash/status/initial/type20USD1/fit Count of genuine cash items which are fit for recycling.
unit1/cash/status/initial/type20USD1/unfit Count of genuine cash items which are unfit for recycling.
unit1/cash/status/initial/type20USD1/suspect Count of suspected counterfeit cash items.
unit1/cash/status/initial/type20USD1/counterfeit Count of counterfeit cash items.
unit1/cash/status/initial/type20USD1/inked Count of cash items which have been identified as ink stained.

Properties
unit1/cash/status/out The items moved from this storage unit by cash commands to another destination since the last replenishment of this unit. This includes intermediate positions such as a stacker, where an item has been moved before moving to the final destination such as another storage unit or presentation to a customer. Counts for non-intermediate positions are reset if initial is set for this unit by Storage.GetStorage. See descriptions for the counts which will not be reset by this command. Intermediate position counts are reset when the intermediate position is empty: • If it is known where the items moved to then the appropriate count or counts are modified. • If it is not known where the items moved, for example because they have been removed manually after jam clearance, then *unknown* is modified.
unit1/cash/status/out/presented The items dispensed from this storage unit which are or were customer accessible.
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.
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.
unit1/cash/status/out/unknown The items dispensed from this storage unit which moved to an unknown position.
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. This is not reset if initial is set for this unit by Storage.GetStorage.
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. This is not reset if initial is set for this unit by Storage.GetStorage.
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. This is not reset if initial is set for this unit by Storage.GetStorage.
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. This also reports items in the *transport*, where an item has jammed before being deposited in the storage unit. Counts other than *transport* are reset if initial is set for this unit by Storage.GetStorage. See descriptions for the counts which will not be reset by this command. The *transport* count is reset when it is empty: • If it is known where the items moved to then the appropriate count or counts are modified. • If it is not known where the items moved, for example because they have been removed manually after jam clearance, then *unknown* is modified.
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.
unit1/cash/status/in/deposited The items deposited in the storage unit during a Cash In transaction.

Properties
unit1/cash/status/in/retracted The items retracted into 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*.
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.
unit1/cash/status/in/distributed The items deposited in this storage unit originating from another storage unit but not rejected.
unit1/cash/status/in/transport The items which were intended to be deposited in this storage unit but are not yet deposited. Typical use case for this property is tracking items after a jam during CashAcceptor.CashInEnd. This is not reset if initial is set for this unit by Storage.GetStorage.
unit1/cash/status/accuracy Describes the accuracy of the counts reported by *out* and *in*. 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.
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. This is based on hardware detection, either on sensors or counts. • `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. If the storage unit has hardwareSensors, this state is not set by counts.
unit1/cash/status/operationStatus On some devices it may be possible to allow items to be dispensed in a recycling storage unit while deposit is inoperable or vice-versa. This property allows the Service to report that one operation is possible while the other is not, without taking the storage unit out of Service completely with status or replenishmentStatus. Following values are possible: • `dispenseInoperative` - Dispense operations are possible and deposit operations are not possible on this recycling storage unit. • `depositInoperative` - Deposit operations are possible and dispense operations are not possible on this recycling storage unit. If not reported, *status* and *replenishmentStatus* apply to both cash out and cash in operations.
unit1/card The card related contents, status and configuration of the unit.
unit1/card/capabilities Indicates the card storage unit capabilities.

Properties
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.
unit1/card/capabilities/hardwareSensors Indicates whether the storage unit has hardware sensors that can detect threshold states. default: false
unit1/card/configuration Indicates the card storage unit configuration.
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.
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`
unit1/card/status Indicates the card storage unit status.
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`
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`

Properties

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
```

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.

20.3.2 Storage.StorageThresholdEvent

This 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.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
"unit1": {	object	
"id": "RC1",	string	
"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": ["type20USD1", "type50USD1"]	array (string)	
},		
"configuration": {	object	
"types": {	object	
See types properties.		
},		

Payload (version 1.0)	Type	Required
"items": {	object	
See items properties.		
},		
"currency": "USD",	string	
"value": 20,	number	
"highThreshold": 500,	integer	
"lowThreshold": 10,	integer	
"appLockIn": false,	boolean	
"appLockOut": false,	boolean	
"cashItems": ["type20USD1", "type50USD1"],	array (string)	
"name": "$10",	string	
"maxRetracts": 5	integer	
},		
"status": {	object	
"index": 4,	integer	
"initial": {	object	
"unrecognized": 0,	integer	
"*type20USD1*": {	object	
"fit": 0,	integer	
"unfit": 0,	integer	
"suspect": 0,	integer	
"counterfeit": 0,	integer	
"inked": 0	integer	
},		
"*type50USD1": {*	object	
See type20USD1 properties.		
}		
},		
"out": {	object	
"presented": {	object	
See initial properties.		
},		
"rejected": {	object	
See initial properties.		
},		
"distributed": {	object	
See initial properties.		
},		
"unknown": {	object	
See initial properties.		
},		

Payload (version 1.0)	Type	Required
"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	
See initial properties.		
},		
"transport": {	object	
See initial properties.		
}		
},		
"accuracy": "notSupported",	string	
"replenishmentStatus": "ok",	string	
"operationStatus": "dispenseInoperative"	string	
}		
},		
"card": {	object	
"capabilities": {	object	
"type": "retain",	string	
"hardwareSensors": false	boolean	
},		
"configuration": {	object	
"cardID": "LoyaltyCard",	string	
"threshold": 0	integer	

Payload (version 1.0)	Type	Required
},		
"status": {	object	
"initialCount": 0,	integer	
"count": 0,	integer	
"retainCount": 0,	integer	
"replenishmentStatus": "ok"	string	
}		
}		
}		
}		

Properties

unit1 (example name)

The object contains a single storage unit.

Property name constraints:

```
pattern: ^unit[0-9A-Za-z]+$
```

unit1/id

An identifier which can be used for cUnitID in CDM/CIM XFS 3.x migration. Not required if not applicable.

Property value constraints:

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

unit1/positionName

Fixed physical name for the position.

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
```

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.

unit1/serialNumber

The storage unit's serial number if it can be read electronically.

unit1/cash

The cash related contents, status and configuration of the unit.

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.

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.

Properties
unit1/cash/capabilities/types/cashIn The unit can accept cash items. If *cashOut* is also true then the unit can recycle.
unit1/cash/capabilities/types/cashOut The unit can dispense cash items. If *cashIn* is also true then the unit can recycle.
unit1/cash/capabilities/types/replenishment Replenishment container. A storage unit can be refilled from or emptied to a replenishment container.
unit1/cash/capabilities/types/cashInRetract Retract unit. Items can be retracted into this unit during Cash In operations.
unit1/cash/capabilities/types/cashOutRetract Retract unit. Items can be retracted into this unit during Cash Out operations.
unit1/cash/capabilities/types/reject Reject unit. Items can be rejected into this unit.
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.
unit1/cash/capabilities/items/fit The storage unit can store cash items which are fit for recycling.
unit1/cash/capabilities/items/unfit The storage unit can store cash items which are unfit for recycling.
unit1/cash/capabilities/items/unrecognized The storage unit can store unrecognized cash items.
unit1/cash/capabilities/items/counterfeit The storage unit can store counterfeit cash items.
unit1/cash/capabilities/items/suspect The storage unit can store suspect counterfeit cash items.
unit1/cash/capabilities/items/inked The storage unit can store cash items which have been identified as ink stained.
unit1/cash/capabilities/items/coupon Storage unit containing coupons or advertising material.
unit1/cash/capabilities/items/document Storage unit containing documents.
unit1/cash/capabilities/hardwareSensors Indicates whether the storage unit has sensors which report the status. If true, then hardware sensors will override count-based replenishment status for *empty* and *full*. Other replenishment states can be overridden by counts.
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`
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.

Properties
unit1/cash/capabilities/cashItems An array containing multiple cash items, listing what a storage unit is physically capable of handling. For example a coin storage unit may be restricted to one coin denomination due to the hardware. Each member is the object name of a cash item reported by CashManagement.GetBankNoteTypes.
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.
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.
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.
unit1/cash/configuration/currency ISO 4217 currency identifier [Ref. cashmanagement-1]. May only be modified in an exchange state if applicable. May be omitted if the unit is configured to store mixed currencies or non-cash items. Property value constraints: `pattern: ^[A-Z]{3}$`
unit1/cash/configuration/value Absolute value of a cash item or items. 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. If applied to a storage unit, this applies to all contents, may be 0 if mixed and may only be modified in an exchange state if applicable. Property value constraints: `minimum: 0`
unit1/cash/configuration/highThreshold If specified, replenishmentStatusis set to *high* if the total number of items in the storage unit is greater than this number. The total number is not reported directly, but derived from *initial + in - out*. If not specified, high is based on hardware sensors if supported - see hardwareSensors. Property value constraints: `minimum: 1`
unit1/cash/configuration/lowThreshold If specified, replenishmentStatusis set to *low* if total number of items in the storage unit is less than this number. The total number is not reported directly, but derived from *initial + in - out*. If not specified, low is based on hardware sensors if supported - see hardwareSensors. Property value constraints: `minimum: 1`
unit1/cash/configuration/appLockIn If true, items cannot be accepted into the storage unit in Cash In operations.
unit1/cash/configuration/appLockOut If true, items cannot be dispensed from the storage unit in Cash Out operations.
unit1/cash/configuration/cashItems An array containing multiple cash items, listing what a storage unit is capable of or configured to handle. Each member is the object name of a cash item reported by CashManagement.GetBankNoteTypes.
unit1/cash/configuration/name Application configured name of the unit.

Properties
unit1/cash/configuration/maxRetracts If specified, this is the number of retract operations allowed into the unit. Only applies to retract units. If retractOperationsequals this number, then no further retracts are allowed into this storage unit. If not specified, the maximum number is not limited by counts. Property value constraints: `minimum: 1`
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. Note that the count of items in the storage unit must be derived from the counts reported.
unit1/cash/status/index Assigned by the Service. Will be a unique number which can be used to determine *usNumber* in XFS 3.x migration. This can change as storage units are added and removed from the storage collection. Property value constraints: `minimum: 1`
unit1/cash/status/initial The cash related items which were in the storage unit at the last replenishment.
unit1/cash/status/initial/unrecognized Count of unrecognized items handled by the cash interface.
unit1/cash/status/initial/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification.
unit1/cash/status/initial/type20USD1/fit Count of genuine cash items which are fit for recycling.
unit1/cash/status/initial/type20USD1/unfit Count of genuine cash items which are unfit for recycling.
unit1/cash/status/initial/type20USD1/suspect Count of suspected counterfeit cash items.
unit1/cash/status/initial/type20USD1/counterfeit Count of counterfeit cash items.
unit1/cash/status/initial/type20USD1/inked Count of cash items which have been identified as ink stained.
unit1/cash/status/out The items moved from this storage unit by cash commands to another destination since the last replenishment of this unit. This includes intermediate positions such as a stacker, where an item has been moved before moving to the final destination such as another storage unit or presentation to a customer. Counts for non-intermediate positions are reset if initial is set for this unit by Storage.GetStorage. See descriptions for the counts which will not be reset by this command. Intermediate position counts are reset when the intermediate position is empty: • If it is known where the items moved to then the appropriate count or counts are modified. • If it is not known where the items moved, for example because they have been removed manually after jam clearance, then *unknown* is modified.
unit1/cash/status/out/presented The items dispensed from this storage unit which are or were customer accessible.
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.

Properties
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.
unit1/cash/status/out/unknown The items dispensed from this storage unit which moved to an unknown position.
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. This is not reset if initial is set for this unit by Storage.GetStorage.
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. This is not reset if initial is set for this unit by Storage.GetStorage.
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. This is not reset if initial is set for this unit by Storage.GetStorage.
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. This also reports items in the *transport*, where an item has jammed before being deposited in the storage unit. Counts other than *transport* are reset if initial is set for this unit by Storage.GetStorage. See descriptions for the counts which will not be reset by this command. The *transport* count is reset when it is empty: • If it is known where the items moved to then the appropriate count or counts are modified. • If it is not known where the items moved, for example because they have been removed manually after jam clearance, then *unknown* is modified.
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.
unit1/cash/status/in/deposited The items deposited in the storage unit during a Cash In transaction.
unit1/cash/status/in/retracted The items retracted into 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*.
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.
unit1/cash/status/in/distributed The items deposited in this storage unit originating from another storage unit but not rejected.
unit1/cash/status/in/transport The items which were intended to be deposited in this storage unit but are not yet deposited. Typical use case for this property is tracking items after a jam during CashAcceptor.CashInEnd. This is not reset if initial is set for this unit by Storage.GetStorage.

Properties
unit1/cash/status/accuracy Describes the accuracy of the counts reported by *out* and *in*. 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.
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. This is based on hardware detection, either on sensors or counts. • `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. If the storage unit has hardwareSensors, this state is not set by counts.
unit1/cash/status/operationStatus On some devices it may be possible to allow items to be dispensed in a recycling storage unit while deposit is inoperable or vice-versa. This property allows the Service to report that one operation is possible while the other is not, without taking the storage unit out of Service completely with status or replenishmentStatus. Following values are possible: • `dispenseInoperative` - Dispense operations are possible and deposit operations are not possible on this recycling storage unit. • `depositInoperative` - Deposit operations are possible and dispense operations are not possible on this recycling storage unit. If not reported, *status* and *replenishmentStatus* apply to both cash out and cash in operations.
unit1/card The card related contents, status and configuration of the unit.
unit1/card/capabilities Indicates the card storage unit capabilities.
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.
unit1/card/capabilities/hardwareSensors Indicates whether the storage unit has hardware sensors that can detect threshold states. default: false
unit1/card/configuration Indicates the card storage unit configuration.
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.

Properties

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
```

unit1/card/status

Indicates the card storage unit status.

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
```

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
```

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
```

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.

20.3.3 Storage.StorageErrorEvent

This event is generated if there is a problem with a storage unit during the execution of a command.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
"failure": "empty",	string	
"unit":	undefined	
"*unit1*": {	object	
"id": "RC1",	string	
"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": ["type20USD1", "type50USD1"]	array (string)	
},		
"configuration": {	object	
"types": {	object	
See types properties.		

Payload (version 1.0)	Type	Required
},		
"items": {	object	
See items properties.		
},		
"currency": "USD",	string	
"value": 20,	number	
"highThreshold": 500,	integer	
"lowThreshold": 10,	integer	
"appLockIn": false,	boolean	
"appLockOut": false,	boolean	
"cashItems": ["type20USD1", "type50USD1"],	array (string)	
"name": "$10",	string	
"maxRetracts": 5	integer	
},		
"status": {	object	
"index": 4,	integer	
"initial": {	object	
"unrecognized": 0,	integer	
"*type20USD1*": {	object	
"fit": 0,	integer	
"unfit": 0,	integer	
"suspect": 0,	integer	
"counterfeit": 0,	integer	
"inked": 0	integer	
},		
"*type50USD1": {*	object	
See type20USD1 properties.		
}		
},		
"out": {	object	
"presented": {	object	
See initial properties.		
},		
"rejected": {	object	
See initial properties.		
},		
"distributed": {	object	
See initial properties.		
},		
"unknown": {	object	
See initial properties.		

Payload (version 1.0)	Type	Required
},		
"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	
See initial properties.		
},		
"transport": {	object	
See initial properties.		
}		
},		
"accuracy": "notSupported",	string	
"replenishmentStatus": "ok",	string	
"operationStatus": "dispenseInoperative"	string	
}		
},		
"card": {	object	
"capabilities": {	object	
"type": "retain",	string	
"hardwareSensors": false	boolean	
},		
"configuration": {	object	
"cardID": "LoyaltyCard",	string	

Payload (version 1.0)	Type	Required
"threshold": 0	integer	
},		
"status": {	object	
"initialCount": 0,	integer	
"count": 0,	integer	
"retainCount": 0,	integer	
"replenishmentStatus": "ok"	string	
}		
}		
}		
}		

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/unit1 (example name)

The object contains a single storage unit.

Property name constraints:

```
pattern: ^unit[0-9A-Za-z]+$
```

unit/unit1/id

An identifier which can be used for cUnitID in CDM/CIM XFS 3.x migration. Not required if not applicable.

Property value constraints:

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

unit/unit1/positionName

Fixed physical name for the position.

unit/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/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.

Properties
unit/unit1/serialNumber The storage unit's serial number if it can be read electronically.
unit/unit1/cash The cash related contents, status and configuration of the unit.
unit/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/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.
unit/unit1/cash/capabilities/types/cashIn The unit can accept cash items. If *cashOut* is also true then the unit can recycle.
unit/unit1/cash/capabilities/types/cashOut The unit can dispense cash items. If *cashIn* is also true then the unit can recycle.
unit/unit1/cash/capabilities/types/replenishment Replenishment container. A storage unit can be refilled from or emptied to a replenishment container.
unit/unit1/cash/capabilities/types/cashInRetract Retract unit. Items can be retracted into this unit during Cash In operations.
unit/unit1/cash/capabilities/types/cashOutRetract Retract unit. Items can be retracted into this unit during Cash Out operations.
unit/unit1/cash/capabilities/types/reject Reject unit. Items can be rejected into this unit.
unit/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/unit1/cash/capabilities/items/fit The storage unit can store cash items which are fit for recycling.
unit/unit1/cash/capabilities/items/unfit The storage unit can store cash items which are unfit for recycling.
unit/unit1/cash/capabilities/items/unrecognized The storage unit can store unrecognized cash items.
unit/unit1/cash/capabilities/items/counterfeit The storage unit can store counterfeit cash items.
unit/unit1/cash/capabilities/items/suspect The storage unit can store suspect counterfeit cash items.
unit/unit1/cash/capabilities/items/inked The storage unit can store cash items which have been identified as ink stained.
unit/unit1/cash/capabilities/items/coupon Storage unit containing coupons or advertising material.
unit/unit1/cash/capabilities/items/document Storage unit containing documents.
unit/unit1/cash/capabilities/hardwareSensors Indicates whether the storage unit has sensors which report the status. If true, then hardware sensors will override count-based replenishment status for *empty* and *full*. Other replenishment states can be overridden by counts.

Properties
unit/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`
unit/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/unit1/cash/capabilities/cashItems An array containing multiple cash items, listing what a storage unit is physically capable of handling. For example a coin storage unit may be restricted to one coin denomination due to the hardware. Each member is the object name of a cash item reported by CashManagement.GetBankNoteTypes.
unit/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/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/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.
unit/unit1/cash/configuration/currency ISO 4217 currency identifier [Ref. cashmanagement-1]. May only be modified in an exchange state if applicable. May be omitted if the unit is configured to store mixed currencies or non-cash items. Property value constraints: `pattern: ^[A-Z]{3}$`
unit/unit1/cash/configuration/value Absolute value of a cash item or items. 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. If applied to a storage unit, this applies to all contents, may be 0 if mixed and may only be modified in an exchange state if applicable. Property value constraints: `minimum: 0`
unit/unit1/cash/configuration/highThreshold If specified, replenishmentStatusis set to *high* if the total number of items in the storage unit is greater than this number. The total number is not reported directly, but derived from *initial + in - out*. If not specified, high is based on hardware sensors if supported - see hardwareSensors. Property value constraints: `minimum: 1`
unit/unit1/cash/configuration/lowThreshold If specified, replenishmentStatusis set to *low* if total number of items in the storage unit is less than this number. The total number is not reported directly, but derived from *initial + in - out*. If not specified, low is based on hardware sensors if supported - see hardwareSensors. Property value constraints: `minimum: 1`

Properties
unit/unit1/cash/configuration/appLockIn If true, items cannot be accepted into the storage unit in Cash In operations.
unit/unit1/cash/configuration/appLockOut If true, items cannot be dispensed from the storage unit in Cash Out operations.
unit/unit1/cash/configuration/cashItems An array containing multiple cash items, listing what a storage unit is capable of or configured to handle. Each member is the object name of a cash item reported by CashManagement.GetBankNoteTypes.
unit/unit1/cash/configuration/name Application configured name of the unit.
unit/unit1/cash/configuration/maxRetracts If specified, this is the number of retract operations allowed into the unit. Only applies to retract units. If retractOperationsequals this number, then no further retracts are allowed into this storage unit. If not specified, the maximum number is not limited by counts. Property value constraints: `minimum: 1`
unit/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. Note that the count of items in the storage unit must be derived from the counts reported.
unit/unit1/cash/status/index Assigned by the Service. Will be a unique number which can be used to determine *usNumber* in XFS 3.x migration. This can change as storage units are added and removed from the storage collection. Property value constraints: `minimum: 1`
unit/unit1/cash/status/initial The cash related items which were in the storage unit at the last replenishment.
unit/unit1/cash/status/initial/unrecognized Count of unrecognized items handled by the cash interface.
unit/unit1/cash/status/initial/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification.
unit/unit1/cash/status/initial/type20USD1/fit Count of genuine cash items which are fit for recycling.
unit/unit1/cash/status/initial/type20USD1/unfit Count of genuine cash items which are unfit for recycling.
unit/unit1/cash/status/initial/type20USD1/suspect Count of suspected counterfeit cash items.
unit/unit1/cash/status/initial/type20USD1/counterfeit Count of counterfeit cash items.
unit/unit1/cash/status/initial/type20USD1/inked Count of cash items which have been identified as ink stained.

Properties
unit/unit1/cash/status/out The items moved from this storage unit by cash commands to another destination since the last replenishment of this unit. This includes intermediate positions such as a stacker, where an item has been moved before moving to the final destination such as another storage unit or presentation to a customer. Counts for non-intermediate positions are reset if initial is set for this unit by Storage.GetStorage. See descriptions for the counts which will not be reset by this command. Intermediate position counts are reset when the intermediate position is empty: • If it is known where the items moved to then the appropriate count or counts are modified. • If it is not known where the items moved, for example because they have been removed manually after jam clearance, then *unknown* is modified.
unit/unit1/cash/status/out/presented The items dispensed from this storage unit which are or were customer accessible.
unit/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/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/unit1/cash/status/out/unknown The items dispensed from this storage unit which moved to an unknown position.
unit/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. This is not reset if initial is set for this unit by Storage.GetStorage.
unit/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. This is not reset if initial is set for this unit by Storage.GetStorage.
unit/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. This is not reset if initial is set for this unit by Storage.GetStorage.
unit/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. This also reports items in the *transport*, where an item has jammed before being deposited in the storage unit. Counts other than *transport* are reset if initial is set for this unit by Storage.GetStorage. See descriptions for the counts which will not be reset by this command. The *transport* count is reset when it is empty: • If it is known where the items moved to then the appropriate count or counts are modified. • If it is not known where the items moved, for example because they have been removed manually after jam clearance, then *unknown* is modified.
unit/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/unit1/cash/status/in/deposited The items deposited in the storage unit during a Cash In transaction.

Properties
unit/unit1/cash/status/in/retracted The items retracted into 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/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/unit1/cash/status/in/distributed The items deposited in this storage unit originating from another storage unit but not rejected.
unit/unit1/cash/status/in/transport The items which were intended to be deposited in this storage unit but are not yet deposited. Typical use case for this property is tracking items after a jam during CashAcceptor.CashInEnd. This is not reset if initial is set for this unit by Storage.GetStorage.
unit/unit1/cash/status/accuracy Describes the accuracy of the counts reported by *out* and *in*. 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/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. This is based on hardware detection, either on sensors or counts. • `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. If the storage unit has hardwareSensors, this state is not set by counts.
unit/unit1/cash/status/operationStatus On some devices it may be possible to allow items to be dispensed in a recycling storage unit while deposit is inoperable or vice-versa. This property allows the Service to report that one operation is possible while the other is not, without taking the storage unit out of Service completely with status or replenishmentStatus. Following values are possible: • `dispenseInoperative` - Dispense operations are possible and deposit operations are not possible on this recycling storage unit. • `depositInoperative` - Deposit operations are possible and dispense operations are not possible on this recycling storage unit. If not reported, *status* and *replenishmentStatus* apply to both cash out and cash in operations.
unit/unit1/card The card related contents, status and configuration of the unit.
unit/unit1/card/capabilities Indicates the card storage unit capabilities.

Properties
unit/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/unit1/card/capabilities/hardwareSensors Indicates whether the storage unit has hardware sensors that can detect threshold states. default: false
unit/unit1/card/configuration Indicates the card storage unit configuration.
unit/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/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/unit1/card/status Indicates the card storage unit status.
unit/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/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`

Properties
unit/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/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.

21. Vendor Mode Interface

This chapter defines the Vendor Mode interface functionality and messages.

This interface allows for the coordination of access to resources, and should be read in conjunction with the Vendor Application interface.

21.1 General Information

21.1.1 Vendor Mode

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 requests to enter and exit Vendor Mode are synchronized.

Entry into, or exit from, Vendor Mode can be initiated either by an application or by the Vendor Mode Service 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 not required. 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 other Services (except for specific Services which explicitly allow sessions to remain open) as soon as possible and then issue an VendorApplication.EnterModeAcknowledge command to the Vendor Mode service when ready. Once all applications have acknowledged, the Vendor Mode Service 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 VendorApplication.ExitModeAcknowledge 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.

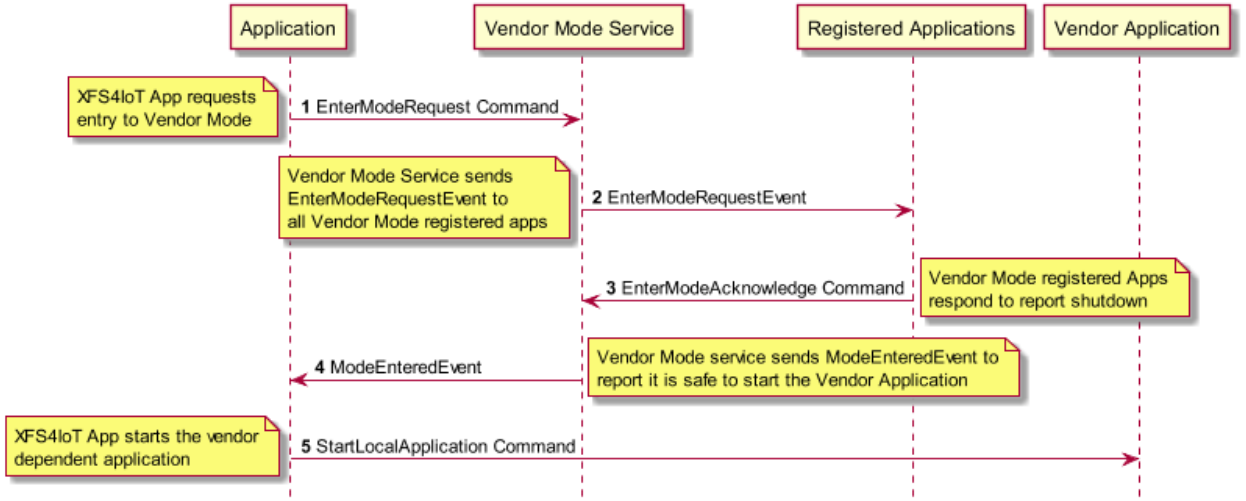
The Vendor Mode Service is used in conjunction with the Vendor Application service. The Vendor Mode Service is responsible for coordinating the release of resources from other services, while the Vendor Application service is responsible for starting the vendor dependent application. The VendorApplication.StartLocalApplication command is used for the latter.

With regard to how the application relates to other services the following rules apply:

1. If the vendor dependent application is published on the same service as a device, then the application 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.

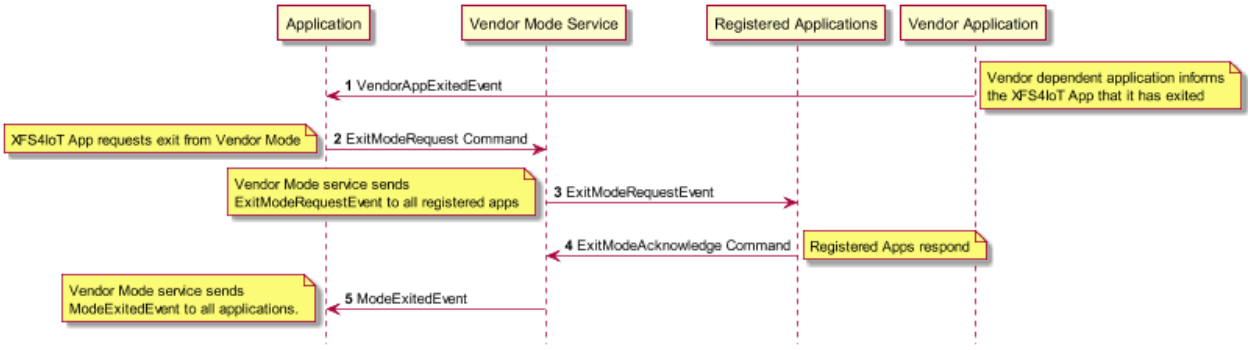
Vendor Mode Entry triggered by an XFS Application



1. An XFS4IoT application calls VendorMode.EnterModeRequest 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 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.

Vendor Mode Entry triggered by an XFS Application



1. The vendor dependent application exits.
2. The XFS4IoT application calls VendorMode.ExitModeRequest to request exit from Vendor Mode.
3. The Vendor Mode Service sends a VendorMode.ExitModeRequestEvent to all applications which have registered to participate in Vendor Mode.
4. The other applications call VendorMode.ExitModeAcknowledge.
5. When all registered applications have reported that they have exited Vendor Mode the Service sends a VendorMode.ModeExitedEvent to report exit from Vendor Mode.

The system is no longer in Vendor Mode.

21.2 Command Messages

21.2.1 VendorMode.Register

This command is issued by an application to register that it wants to participate in Vendor Mode.

This command can be used while in Vendor Mode.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"appName": "ACME Monitoring app"	string	
}		

Properties
timeout Timeout in milliseconds for the command to complete. If set to 0, 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.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

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

Event Messages

None

21.2.2 VendorMode.EnterModeRequest

This command is issued by an application to indicate a logical request to enter Vendor Mode. The Service 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 will change to *enterPending* on receipt of this command and will prevail until all applications have acknowledged, at which time *service* will change to *active* and the command completes. If the command fails when *service* is *enterPending*, *service* is changed to *inactive* and VendorMode.ModeExitedEvent is sent to all registered applications.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

Properties

completionCode
The completion code.

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

Event Messages

None

21.2.3 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 Vendor Mode 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 (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

Properties

completionCode
The completion code.

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

Event Messages

None

21.2.4 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.

This command can be used while in Vendor Mode.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	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

21.2.5 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.

This command can be used while in Vendor Mode.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

Properties

completionCode
The completion code.

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

Event Messages

None

21.3 Unsolicited Messages

21.3.1 VendorMode.EnterModeRequestEvent

This service event is used to indicate the request to enter Vendor Mode.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
}		

21.3.2 VendorMode.ExitModeRequestEvent

This service event is used to indicate the request to exit Vendor Mode.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
}		

21.3.3 VendorMode.ModeEnteredEvent

This event is used to indicate that the system has entered Vendor Mode.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
}		

21.3.4 VendorMode.ModeExitedEvent

This event is used to indicate that the system has exited Vendor Mode.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
"connectedApplications": ["Application1", "Application2"]	array (string)	
}		

Properties

connectedApplications
List of applications that have not shut down.

22. Vendor Application Interface

This chapter defines and describes the functionality of the Vendor Application Service, which is used to start a local application, and set the active interface.

22.1 General Information

22.1.1 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 Vendor Mode 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 Vendor Mode service and the VendorApplication.StartLocalApplication command refer to the Vendor Mode Service documentation, which describes this fully.

The Vendor Mode 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 Vendor Mode Service has been called the VendorApplication.StartLocalApplication command can be used to run the vendor dependent application. When the vendor dependent application exits it sends a VendorApplication.VendorAppExitedEvent event to the main application to indicate that it has exited, the application can then use the Vendor Mode Service to communicate to other services that it is safe to regain control of the hardware.

The VendorApplication.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. VendorApplication.GetActiveInterface reports the currently active interface. Note that the interface can also be changed while the vendor dependent application is running.

22.2 Command Messages

22.2.1 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"appName": "Add example",	string	
"accessLevel": "basic"	string	
}		

Properties

timeout

Timeout in milliseconds for the command to complete. If set to 0, 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:

- `basic` - 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.
- `intermediate` - 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.
- `full` - 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 (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

Properties

completionCode

The completion code.

errorDescription

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

Event Messages

None

22.2.2 VendorApplication.GetActiveInterface

This command is used to retrieve the interface that should be used by the vendor dependent application.

Command Message

Payload (version 1.0)	Type	Required
{		
"timeout": 5000	integer	
}		

Properties

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

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available",	string	
"activeInterface": "consumer"	string	Yes
}		

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:

- `consumer` - The consumer interface.
- `operator` - The operator interface.

Event Messages

None

22.2.3 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 (version 1.0)	Type	Required
{		
"timeout": 5000,	integer	
"activeInterface": "consumer"	string	Yes
}		

Properties

timeout

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

default: 0

activeInterface

Specifies the active interface as one of the following values:

- `consumer` - The consumer interface.
- `operator` - The operator interface.

Completion Message

Payload (version 1.0)	Type	Required
{		
"completionCode": "success",	string	
"errorDescription": "Device not available"	string	
}		

Properties

completionCode

The completion code.

errorDescription

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

Event Messages

- VendorApplication.InterfaceChangedEvent

22.3 Unsolicited Messages

22.3.1 VendorApplication.VendorAppExitedEvent

This event is used to indicate the vendor dependent application has exited, allowing an application the opportunity to exit Vendor Mode.

Unsolicited Message

Payload (version 1.0)	Type	Required
{		
}		

22.3.2 VendorApplication.InterfaceChangedEvent

This event is used to indicate that the required interface has changed. This can be as a result of a VendorApplication.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.

Unsolicited Message

Payload (version 1.0)	Type	Required
`{`		
`"activeInterface": "consumer"`	string	Yes
`}`		

Properties

activeInterface

Specifies the active interface as one of the following values:

- `consumer` - The consumer interface.
- `operator` - The operator interface.