Extensions for Financial Services (XFS)
XFS4IoT Specification
Release 2023-02

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

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

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

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

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

Warning

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

Recipients should notify the committee of any relevant patent rights of which they are aware and to provide supporting documentation.

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

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

© 2023 CEN All rights of exploitation in any form and by any means reserved worldwide for CEN national Members.

Table of Contents

1. Foreword

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
api-4	JSON Schema 2020-12 (https://json-schema.org)

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 message 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 defined using JSON Schema 2020-12 Ref. api-4.

2.3.1 Message Definition

All messages follow the same JSON structure consisting of the following properties:

Property	Property Type	Required
header	object	✓
payload	object, null	

As illustrated in the example below.

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

2.3.2 Header Definition

The headers contains properties common to all messages as well as properties specific to a message type.

Additional properties are not allowed.

Property	Property Type	Required
type	string	✓
name	string	✓
version	string	✓
requestId	integer	
timeout	integer	
status	string, null	
completionCode	string, null	
errorDescription	string, null	

The following example illustrates the header for a *Common.Status* command message.

```
{
    "header": {
        "type": "command",
        "name": "Common.Status",
        "version": "1.0",
        "requestId": 12345,
        "timeout": 1000
    }
}
```

type Property

The message type.

name Property

The message name, for example *Common.Status*.

version Property

The message version, for example *1.0*.

requestId Property

Unique request identifier supplied by the client used to correlate the command message with acknowledge, event and completion messages. The client will supply values that are positive, incremental and greater than or equal to 0. The service will check that the requestId does not conflict with a currently executing or queued command request from the same client and return status *invalidRequestID* if it does.

Unsolicited messages do not have a requestId.

timeout Property

This property is only applicable to *command* messages.

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

Default: 0

status Property

This property is only applicable to *acknowledge* messages.

If null the command has been accepted for execution and will complete with a completion message. Otherwise this property can be one of the following values:

- `invalidMessage` - The JSON in the message is invalid and can't be parsed.
- `invalidRequestID` - The request ID on the command is invalid. This could be because the value is not an integer, has a zero value, or because a command with the same request ID from the same client is already queued or is executing.
- `tooManyRequests` - The service has currently received and queued more requests than it can process.

Default: null

completionCode Property

This property is only applicable to *completion* messages.

If null the command completed successfully. Otherwise, the property will contain one of the following values:

- `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.
- `notEnoughSpace` - There is not enough space on the storage.

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

Default: null

errorDescription Property

This property is only applicable to *completion* messages for which the *completionCode* value is neither *canceled* or *timeOut*.

If not null, this contains additional vendor dependent information to assist with problem resolution. The format of this string should not be relied on.

Default: null

2.3.3 Payload Definition

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

If not null, the payload cannot be empty. It must contain at least one property.

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.

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.

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

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 parsed 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.

If for any reason the service does not accept and queue the command request, the acknowledge message header *status* property will indicate the reason. When this occurs, the acknowledge message is the final message related to the command request.

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.

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.

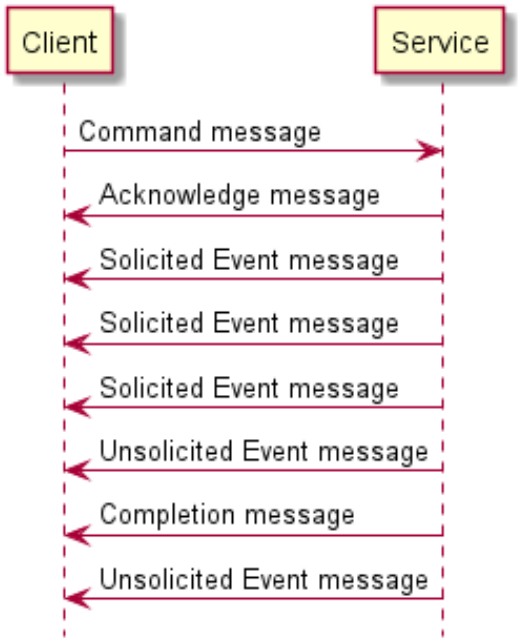
Services may also send unsolicited events directly to the client. This can happen at any time that the service connection is open. This could be during the processing of a command, or between commands.

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.

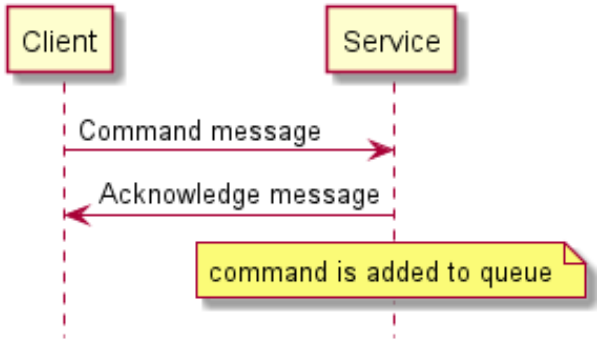
2.5.2 Command Queuing

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*.

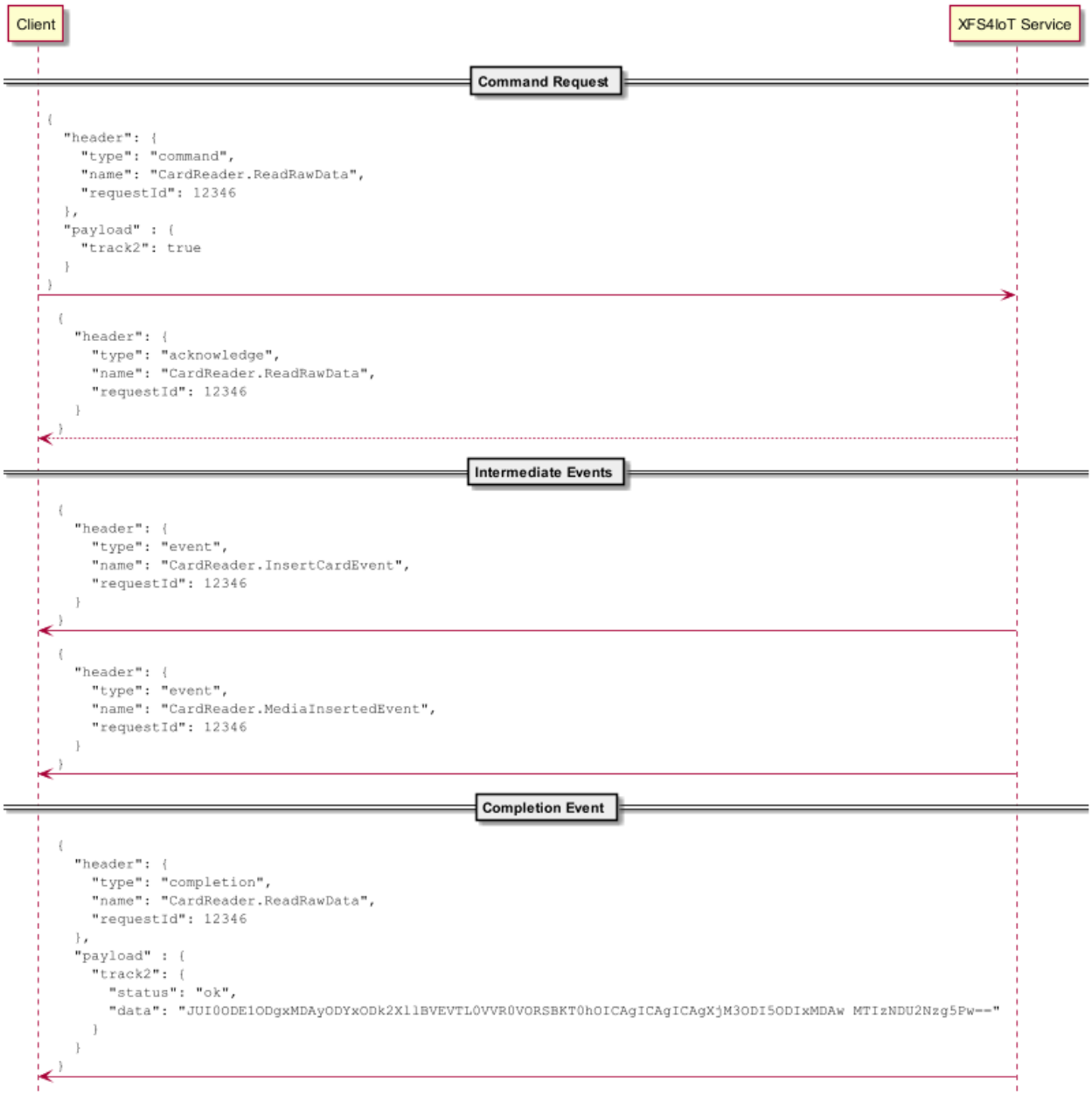
Clients should expect that, at some future point, commands may complete with a *completionCode* other than *canceled*. For example, device state prevents the command canceling forcing it to complete as if no cancel request had been received.

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.

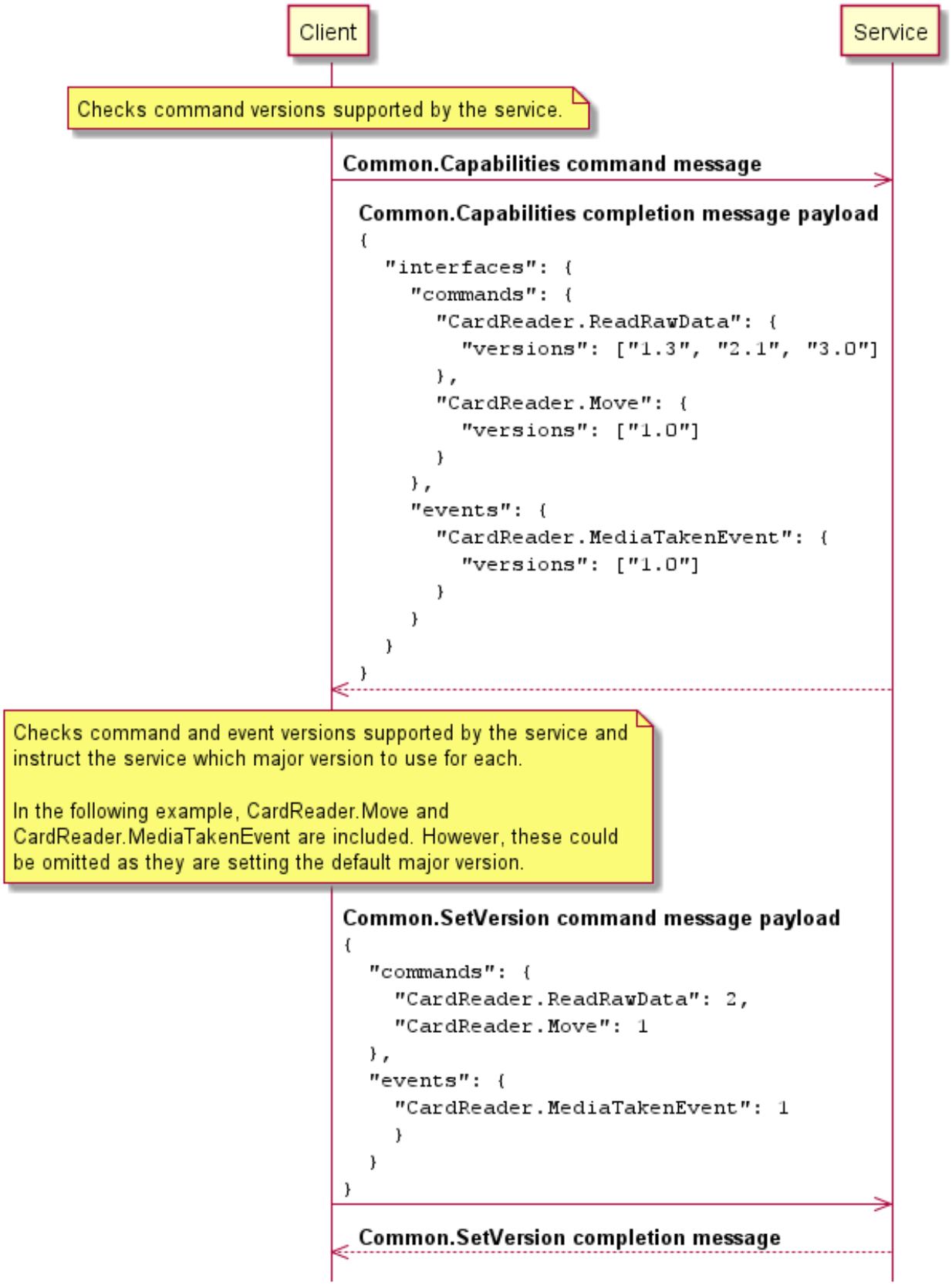
For additional completion, event and unsolicited message properties, clients should expect that new properties may be added and if not required, ignored. That is, clients should not break because they do not recognize additional properties.

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
  1.
- 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

Not all commands that could require E2E security are currently covered. When E2E security is being enforced by a device, sensitive commands with no token defined will be blocked from executing. This is required to avoid any way of bypassing security. For example, the cash *CashDispenser.Dispense* command has a token format defined but the CashDispenser.TestCashUnits command does not. For security, *CashDispenser. TestCashUnits* must be blocked from dispensing cash, otherwise an attacker could simply replace the *CashDispenser. Dispense* with calls to *CashDispenser.TestCashUnits*. Restrictions are documented for each affected command.

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 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"vendorName": "ACME ATM Hardware GmbH",`	string	✓
`"services": [{`	array (object), null	
`"serviceURI": "wss://ATM1:123/xfs4iot/v1.0/CardReader"`	string	✓
`}]`		
`}`		

Properties

vendorName
Freeform string naming the hardware vendor.

services
Array of one or more services exposed by the publisher. This property is null if no services available.
default: null

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

Properties

vendorName
Freeform string naming the hardware vendor.

services
Array of one or more services exposed by the publisher. This property is null if no services available.
default: null

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 2.0)
This message does not define any properties.

Completion Message

Payload (version 2.0)	Type	Required
{		
"common": {	object	✓
"device": "online",	string	✓
"devicePosition": "notInPosition",	string, null	
"powerSaveRecoveryTime": 10,	integer, null	
"antiFraudModule": "ok",	string, null	
"exchange": "active",	string, null	
"endToEndSecurity": "enforced"	string, null	
},		
"cardReader": {	object, null	
"media": "unknown",	string, null	
"security": "notReady",	string, null	
"chipPower": "unknown",	string, null	
"chipModule": "ok",	string, null	
"magWriteModule": "ok",	string, null	
"frontImageModule": "ok",	string, null	
"backImageModule": "ok"	string, null	
},		
"cashAcceptor": {	object, null	
"intermediateStacker": "empty",	string, null	
"stackerItems": "customerAccess",	string, null	
"banknoteReader": "ok",	string, null	
"dropBox": true,	boolean, null	
"positions": [{	array (object), null	
"position": "inLeft",	string	✓
"shutter": "closed",	string, null	
"positionStatus": "empty",	string, null	

Payload (version 2.0)	Type	Required
"transport": "ok",	string, null	
"transportStatus": "empty"	string, null	
}]		
},		
"cashDispenser": {	object, null	
"intermediateStacker": "empty",	string, null	
"positions": [{	array (object), null	
"position": "outDefault",	string	
"shutter": "closed",	string, null	
"positionStatus": "empty",	string, null	
"transport": "ok",	string, null	
"transportStatus": "empty"	string, null	
}]		
},		
"cashManagement": {	object, null	
"dispenser": "ok",	string, null	
"acceptor": "ok"	string, null	
},		
"check": {	object, null	
"acceptor": "ok",	string, null	
"media": "present",	string, null	
"toner": "full",	string, null	
"ink": "full",	string, null	
"frontImageScanner": "ok",	string, null	
"backImageScanner": "ok",	string, null	
"mICRReader": "ok",	string, null	
"stacker": "empty",	string, null	
"rebuncher": "empty",	string, null	
"mediaFeeder": "notEmpty",	string, null	
"positions": {	object, null	
"input": {	object, null	
"shutter": "closed",	string, null	✓
"positionStatus": "empty",	string, null	
"transport": "ok",	string, null	
"transportMediaStatus": "empty",	string, null	
"jammedShutterPosition": "notJammed"	string, null	
},		
"output": See check/positions/input properties	object, null	
"refused": See check/positions/input properties	object, null	

Payload (version 2.0)	Type	Required
}		
},		
"mixedMedia": {	object, null	
"modes": {	object	✓
"cashAccept": true,	boolean, null	
"checkAccept": true	boolean, null	
}		
},		
"keyManagement": {	object, null	
"encryptionState": "ready",	string, null	
"certificateState": "unknown"	string, null	
},		
"keyboard": {	object, null	
"autoBeepMode": {	object	✓
"activeAvailable": false,	boolean, null	
"inactiveAvailable": false	boolean, null	
}		
},		
"textTerminal": {	object, null	
"keyboard": "on",	string, null	
"keyLock": "on",	string, null	
"displaySizeX": 0,	integer, null	
"displaySizeY": 0	integer, null	
},		
"printer": {	object, null	
"media": "unknown",	string, null	
"paper": {	object, null	
"upper": "unknown",	string, null	
"lower": "unknown",	string, null	
"external": "unknown",	string, null	
"aux": "unknown",	string, null	
"aux2": "unknown",	string, null	
"park": "unknown",	string, null	
"*vendorSpecificPaperSupply*": "unknown"	string, null	
},		
"toner": "unknown",	string, null	
"ink": "unknown",	string, null	
"lamp": "unknown",	string, null	
"retractBins": [{	array (object), null	

Payload (version 2.0)	Type	Required
"state": "unknown",	string, null	
"count": 0	integer, null	
}],		
"mediaOnStacker": 7,	integer, null	
"paperType": {	object, null	
"upper": "unknown",	string, null	
"lower": "unknown",	string, null	
"external": "unknown",	string, null	
"aux": "unknown",	string, null	
"aux2": "unknown",	string, null	
"park": "unknown",	string, null	
"*exampleProperty1*": "unknown",	string, null	
"*exampleProperty2*": See *printer/paperType/exampleProperty1*	string, null	
},		
"blackMarkMode": "unknown"	string, null	
},		
"barcodeReader": {	object, null	
"scanner": "on"	string	✓
},		
"biometric": {	object, null	
"subject": "present",	string, null	
"capture": false,	boolean, null	
"dataPersistence": "persist",	string, null	
"remainingStorage": 0	integer, null	
},		
"camera": {	object, null	
"media": {	object, null	
"room": "ok",	string, null	
"person": "ok",	string, null	
"exitSlot": "ok",	string, null	
"*vendorSpecificCameraMedia*": "ok"	string, null	
},		
"cameras": {	object, null	
"room": "ok",	string, null	
"person": "ok",	string, null	
"exitSlot": "ok",	string, null	
"*vendorSpecificCameraState*": See *camera/media/vendorSpecificCameraMedia*	string, null	
},		
"pictures": {	object, null	

Payload (version 2.0)	Type	Required
"room": 0,	integer, null	
"person": 0,	integer, null	
"exitSlot": 0,	integer, null	
"*vendorSpecificCameraPictures*": 0	integer, null	
}		
},		
"lights": {	object, null	
"cardReader": {	object, null	
"position": "left",	string	✓
"flashRate": "off",	string, null	
"color": "red",	string, null	
"direction": "entry"	string, null	
},		
"pinPad": See lights/cardReader properties	object, null	
"notesDispenser": See lights/cardReader properties	object, null	
"coinDispenser": See lights/cardReader properties	object, null	
"receiptPrinter": See lights/cardReader properties	object, null	
"passbookPrinter": See lights/cardReader properties	object, null	
"envelopeDepository": See lights/cardReader properties	object, null	
"checkUnit": See lights/cardReader properties	object, null	
"billAcceptor": See lights/cardReader properties	object, null	
"envelopeDispenser": See lights/cardReader properties	object, null	
"documentPrinter": See lights/cardReader properties	object, null	
"coinAcceptor": See lights/cardReader properties	object, null	
"scanner": See lights/cardReader properties	object, null	
"contactless": See lights/cardReader properties	object, null	
"cardReader2": See lights/cardReader properties	object, null	
"notesDispenser2": See lights/cardReader properties	object, null	
"billAcceptor2": See lights/cardReader properties	object, null	
"statusGood": See lights/cardReader properties	object, null	
"statusWarning": See lights/cardReader properties	object, null	
"statusBad": See lights/cardReader properties	object, null	
"statusSupervisor": See lights/cardReader properties	object, null	
"statusInService": See lights/cardReader properties	object, null	
"fasciaLight": See lights/cardReader properties	object, null	
"*vendorSpecificLight*": See *lights/cardReader* properties	object, null	
},		
"auxiliaries": {	object, null	
"operatorSwitch": "run",	string, null	
"tamperSensor": "on",	string, null	

Payload (version 2.0)	Type	Required
"internalTamperSensor": "on",	string, null	
"seismicSensor": "on",	string, null	
"heatSensor": "on",	string, null	
"proximitySensor": "present",	string, null	
"ambientLightSensor": "veryDark",	string, null	
"enhancedAudioSensor": "present",	string, null	
"bootSwitchSensor": "off",	string, null	
"consumerDisplaySensor": "off",	string, null	
"operatorCallButtonSensor": "off",	string, null	
"handsetSensor": "onTheHook",	string, null	
"headsetMicrophoneSensor": "present",	string, null	
"fasciaMicrophoneSensor": "off",	string, null	
"safeDoor": "closed",	string, null	
"vandalShield": "closed",	string, null	
"cabinetFrontDoor": "closed",	string, null	
"cabinetRearDoor": "closed",	string, null	
"cabinetLeftDoor": "closed",	string, null	
"cabinetRightDoor": "closed",	string, null	
"openClosedIndicator": "closed",	string, null	
"audio": {	object, null	
"rate": "on",	string, null	
"signal": "keypress"	string, null	
},		
"heating": "off",	string, null	
"consumerDisplayBacklight": "off",	string, null	
"signageDisplay": "off",	string, null	
"volume": 1,	integer, null	
"UPS": {	object, null	
"low": true,	boolean, null	
"engaged": false,	boolean, null	
"powering": false,	boolean, null	
"recovered": false	boolean, null	
},		
"audibleAlarm": "on",	string, null	
"enhancedAudioControl": "publicAudioManual",	string, null	
"enhancedMicrophoneControl": "publicAudioManual",	string, null	
"microphoneVolume": 1	integer, null	
},		
"vendorMode": {	object, null	
"device": "online",	string, null	

Payload (version 2.0)	Type	Required
"service": "enterPending"	string, null	
},		
"vendorApplication": {	object, null	
"accessLevel": "notActive"	string	✓
}		
}		

Properties

common

Status information common to all XFS4IoT services.

common/device

Specifies the state of the device. This property is required in Common.Status, but may be null in Common.StatusChangedEvent if it has not changed. 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. This property is null in Common.Status if position status reporting is not supported, otherwise the 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.

default: null

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 the power saving mode has not been activated. This property is null in Common.Status if power save control is not supported.

Property value constraints:

```
minimum: 0
```

default: null

Properties
common/antiFraudModule Specifies the state of the anti-fraud module if available. This property is null in Common.Status if there is no anti-fraud module, otherwise the 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. default: null
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. This property is null in Common.Status if not supported, otherwise the following values are possible: • `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: null
common/endToEndSecurity Specifies the status of end to end security support on this device. This property is null in Common.Status if E2E security is not supported by this hardware and any command can be called without a token, otherwise the following values are possible. Also see Common.CapabilityProperties.endToEndSecurity. • `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: null
cardReader Status information for XFS4IoT services implementing the CardReader interface. This will be null if the CardReader interface is not supported. default: null
cardReader/media Specifies the transport/exit position media state. This property will be null if the capability to report media position is not supported by the device (e.g., a typical swipe reader or contactless chip card reader), otherwise one of the following values: • `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. default: null

Properties
cardReader/security Specifies the state of the security module. This property will be null if no security module is available, otherwise one of the following values: • `notReady` - The security module is not ready to process cards or is inoperable. • `open` - The security module is open and ready to process cards. default: null
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. This property will be null if the capability to report the state of the chip is not supported by the ID card unit device and will apply to contactless chip card readers, otherwise one of the following values: • `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. default: null
cardReader/chipModule Specifies the state of the chip card module reader. This property will be null if reporting the chip card module status is not supported, otherwise one of the following values: • `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. default: null
cardReader/magWriteModule Specifies the state of the magnetic card writer. This property will be null if reporting the magnetic card writing module status is not supported, otherwise one of the following values: • `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. default: null
cardReader/frontImageModule Specifies the state of the front image reader. This property will be null if reporting the front image reading module status is not supported, otherwise one of the following values: • `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. default: null
cardReader/backImageModule Specifies the state of the back image reader. This property will be null if reporting the back image reading module status is not supported, otherwise one of the following values: • `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. default: null

Properties
cashAcceptor Status information for XFS4IoT services implementing the CashAcceptor interface. This will be null if the CashAcceptor interface is not supported. default: null
cashAcceptor/intermediateStacker Supplies the state of the intermediate stacker. This property is null in Common.Status if the physical device has no intermediate stacker, otherwise 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. default: null
cashAcceptor/stackerItems This property informs the application whether items on the intermediate stacker have been in customer access. This property is null in Common.Status if the physical device has no intermediate stacker, otherwise 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. default: null
cashAcceptor/banknoteReader Supplies the state of the banknote reader. This property is null in Common.Status if the physical device has no banknote reader, otherwise 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. default: null
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 property 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. This property may be null if there is no drop box or its state has not changed in Common.StatusChangedEvent. default: null
cashAcceptor/positions Array of structures reporting status for each position from which items can be accepted. This may be null in Common.StatusChangedEvent if no position states have changed. default: null

Properties
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. This property is null in Common.Status if the physical position has no shutter, otherwise 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. default: null
cashAcceptor/positions/positionStatus The status of the input or output position. This property is null in Common.Status if the device is not capable of reporting whether items are at the position, otherwise 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. • `foreignItems` - Foreign items have been detected in the position. default: null
cashAcceptor/positions/transport Supplies the state of the transport mechanism. The transport is defined as any area leading to or from the position. This property is null in Common.Status if the device has no transport or transport state reporting is not supported, otherwise 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. default: null

Properties
cashAcceptor/positions/transportStatus Returns information regarding items which may be on the transport. If the device is a recycler device it is possible that the transport will not be empty due to a previous dispense operation. This property is null in Common.Status if the device has no transport or is not capable of reporting whether items are on the transport, otherwise 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. default: null
cashDispenser Status information for XFS4IoT services implementing the CashDispenser interface. This will be null if the CashDispenser interface is not supported. default: null
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. This property is null in Common.Status if the physical device has no intermediate stacker, otherwise the 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. default: null
cashDispenser/positions Array of structures for each position to which items can be dispensed or presented. This may be null in Common.StatusChangedEvent if no position states have changed. default: null
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. This property is null in Common.Status if the physical position has no shutter, otherwise the 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. default: null
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. This property is null in Common.Status if the device is not capable of reporting whether items are at the position, otherwise 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. default: null
cashDispenser/positions/transport Supplies the state of the transport mechanism. The transport is defined as any area leading to or from the position. This property is null in Common.Status if the device has no transport or transport state reporting is not supported, otherwise 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. default: null
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. This property is null in Common.Status if the device has no transport or is not capable of reporting whether items are on the transport, otherwise 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. default: null
cashManagement Status information for XFS4IoT services implementing the CashManagement interface. This will be null if the CashManagement interface is not supported. default: null

Properties
cashManagement/dispenser Supplies the state of the storage units for dispensing cash. This may be null in Common.Status if the device is not capable of dispensing cash, otherwise the 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. default: null
cashManagement/acceptor Supplies the state of the storage units for accepting cash. This may be null in Common.Status if the device is not capable of accepting cash, otherwise the 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 items are to be automatically retained within storage units (see retainAction), 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. default: null
check Status information for XFS4IoT services implementing the Check interface. This will be null if the Check interface is not supported. default: null
check/acceptor Supplies the state of the overall acceptor storage units. This may be null in Common.StatusChangedEvent if the state has not changed. The following values are possible: • `ok` - All storage units present are in a good state. • `state` - One or more of the storage units is in a high, full or inoperative condition. Items can still be accepted into at least one of the storage units. The status of the storage units can be obtained through the Storage.GetStorage command. • `stop` - Due to a storage unit problem accepting is impossible. No items can be accepted because all of the storage units are in a full or in an inoperative condition. • `unknown` - Due to a hardware error or other condition, the state of the storage units cannot be determined. default: null

Properties
check/media Specifies the state of the media. This may be null in Common.Status if the capability to report the state of the media is not supported by the device, otherwise the following values are possible: • `present` - Media is present in the device. • `notPresent` - Media is not present in the device. • `jammed` - Media is jammed in the device. • `unknown` - The state of the media cannot be determined with the device in its current state. • `position` - Media is at one or more of the input, output and refused positions. default: null
check/toner Specifies the state of the toner or ink supply or the state of the ribbon of the endorser. This may be null in Common.Status if the physical device does not support endorsing or the capability to report the status of the toner/ink is not supported by the device, otherwise the following values are possible: • `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. • `unknown` - Status of toner or ink supply or the ribbon cannot be determined with the device in its current state. default: null
check/ink Specifies the status of the stamping ink in the device. This may be null in Common.Status if the physical device does not support stamping or the capability to report the status of the stamp ink supply is not supported by the device, otherwise the following values are possible: • `full` - Ink supply in the device is full. • `low` - Ink supply in the device is low. • `out` - Ink supply in the device is empty. • `unknown` - Status of the stamping ink supply cannot be determined with the device in its current state. default: null
check/frontImageScanner Specifies the status of the image scanner that captures images of the front of the media items. This may be null in Common.Status if the physical device has no front scanner or the capability to report the status of the front scanner is not supported by the device, otherwise the following values are possible: • `ok` - The front scanner is OK. • `fading` - The front scanner performance is degraded. • `inoperative` - The front scanner is inoperative. • `unknown` - Status of the front scanner cannot be determined with the device in its current state. default: null
check/backImageScanner Specifies the status of the image scanner that captures images of the back of the media items. This may be null in Common.Status if the physical device has no back scanner or the capability to report the status of the back scanner is not supported by the device, otherwise the following values are possible: • `ok` - The back scanner is OK. • `fading` - The back scanner performance is degraded. • `inoperative` - The back scanner is inoperative. • `unknown` - Status of the back scanner cannot be determined with the device in its current state. default: null

Properties
check/mICRReader Specifies the status of the MICR code line reader. This may be null in Common.Status if the physical device has no MICR code line reader or the capability to report the status of the MICR code line reader is not supported by the device, otherwise the following values are possible: • `ok` - The MICR code line reader is OK. • `fading` - The MICR code line reader performance is degraded. • `inoperative` - The MICR code line reader is inoperative. • `unknown` - Status of the MICR code line reader cannot be determined with the device in its current state. default: null
check/stacker Supplies the state of the stacker (also known as an escrow). The stacker is where the media items are held while the application decides what to do with them. This may be null in Common.Status if the physical device has no stacker or the capability to report the status of the stacker is not supported by the device, otherwise the following values are possible: • `empty` - The stacker is empty. • `notEmpty` - The stacker is not empty. • `full` - The stacker is full. This state is set if the number of media items on the stacker has reached maxMediaOnStacker or some physical limit has been reached. • `inoperative` - The stacker is inoperative. • `unknown` - Due to a hardware error or other condition, the state of the stacker cannot be determined. default: null
check/rebuncher Supplies the state of the re-buncher (return stacker). The re-buncher is where media items are re-bunched ready for return to the customer. This may be null in Common.Status if the physical device has no re-buncher or the capability to report the status of the re-buncher is not supported by the device, otherwise the following values are possible: • `empty` - The re-buncher is empty. • `notEmpty` - The re-buncher is not empty. • `full` - The re-buncher is full. This state is set if the number of media items on the re-buncher has reached its physical limit. • `inoperative` - The re-buncher is inoperative. • `unknown` - Due to a hardware error or other condition, the state of the re-buncher cannot be determined. default: null
check/mediaFeeder Supplies the state of the media feeder. This value indicates if there are items on the media feeder waiting for processing via the Check.GetNextItem command. If null, the device has no media feeder or the capability to report the status of the media feeder is not supported by the device. This value can be one of the following values: • `empty` - The media feeder is empty. • `notEmpty` - The media feeder is not empty. • `inoperative` - The media feeder is inoperative. • `unknown` - Due to a hardware error or other condition, the state of the media feeder cannot be determined. default: null
check/positions Specifies the status of the input, output and refused positions. This may be null in Common.StatusChangedEvent if no position states have changed. Property value constraints: `minProperties: 1` default: null

Properties
check/positions/input Specifies the status of the input position. This may be null in Common.StatusChangedEvent if no states have changed for the position. default: null
check/positions/input/shutter Specifies the state of the shutter. This property is null in Common.Status if the physical device has no shutter or shutter state reporting is not supported, otherwise the following values are possible: • `closed` - The shutter is operational and is closed. • `open` - The shutter is operational and is open. • `jammed` - The shutter is jammed and is not operational. • `unknown` - Due to a hardware error or other condition, the state of the shutter cannot be determined. default: null
check/positions/input/positionStatus The status of the position. This property is null in Common.Status if the physical device is not capable of reporting whether or not items are at the position, otherwise 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. default: null
check/positions/input/transport Specifies the state of the transport mechanism. The transport is defined as any area leading to or from the position. This property is null in Common.Status if the physical device has no transport or transport state reporting is not supported, otherwise 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. default: null
check/positions/input/transportMediaStatus Returns information regarding items which may be present on the transport. This property is null in Common.Status if the physical device is not capable of reporting whether or not items are on the transport, otherwise the following values are possible: • `empty` - The transport is empty. • `notEmpty` - The transport is not empty. • `unknown` - Due to a hardware error or other condition it is not known whether there are items on the transport. default: null
check/positions/input/jammedShutterPosition Returns information regarding the position of the jammed shutter. This property is null in Common.Status if the physical device has no shutter or the reporting of the position of a jammed shutter is not supported, otherwise the following values are possible: • `notJammed` - The shutter is not jammed. • `open` - The shutter is jammed, but fully open. • `partiallyOpen` - The shutter is jammed, but partially open. • `closed` - The shutter is jammed, but fully closed. • `unknown` - The position of the shutter is unknown. default: null
check/positions/output Specifies the status of the output position. This may be null in Common.StatusChangedEvent if no states have changed for the position. default: null

Properties
check/positions/refused Specifies the status of the refused position. This may be null in Common.StatusChangedEvent if no states have changed for the position. default: null
mixedMedia Status information for XFS4IoT services implementing the MixedMedia interface. This will be null if the MixedMedia interface is not supported. default: null
mixedMedia/modes Specifies the state of the transaction modes supported by the Service. Property value constraints: `minProperties: 1`
mixedMedia/modes/cashAccept Specifies whether transactions can accept cash. This property may be null if no change required or its state has not changed in Common.StatusChangedEvent. default: null
mixedMedia/modes/checkAccept Specifies whether transactions can accept checks. This property may be null if no change required or its state has not changed in Common.StatusChangedEvent. default: null
keyManagement Status information for XFS4IoT services implementing the KeyManagement interface. This will be null if the KeyManagement interface is not supported. default: null
keyManagement/encryptionState Specifies the state of the encryption module. This may be null in Common.StatusChangedEvent if unchanged. default: null
keyManagement/certificateState Specifies the state of the public verification or encryption key in the PIN certificate modules. This may be null in Common.StatusChangedEvent if unchanged. default: null
keyboard Status information for XFS4IoT services implementing the Keyboard interface. This will be null if the Keyboard interface is not supported. default: null
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. This may be null in Common.StatusChangedEvent if unchanged. default: null
keyboard/autoBeepMode/inactiveAvailable Specifies whether an automatic tone will be generated for all inactive keys. This may be null in Common.StatusChangedEvent if unchanged. default: null

Properties
textTerminal Status information for XFS4IoT services implementing the TextTerminal interface. This will be null if the TextTerminal interface is not supported. default: null
textTerminal/keyboard Specifies the state of the keyboard in the text terminal unit. This property will be null in Common.Status if the keyboard is not available, otherwise one of the following values: • `on` - The keyboard is activated. • `off` - The keyboard is not activated. default: null
textTerminal/keyLock Specifies the state of the keyboard lock of the text terminal unit. This property will be null in Common.Status if the keyboard lock switch is not available, otherwise one of the following values: • `on` - The keyboard lock switch is activated. • `off` - The keyboard lock switch is not activated. default: null
textTerminal/displaySizeX Specifies the horizontal size of the display of the text terminal unit (the number of columns that can be displayed). This property will be null in Common.StatusChangedEvent if unchanged. Property value constraints: `minimum: 0` default: null
textTerminal/displaySizeY Specifies the vertical size of the display of the text terminal unit (the number of rows that can be displayed). This property will be null in Common.StatusChangedEvent if unchanged. Property value constraints: `minimum: 0` default: null
printer Status information for XFS4IoT services implementing the Printer interface. This will be null if the Printer interface is not supported. default: null
printer/media Specifies the state of the print media (i.e. receipt, statement, passbook, etc.) as one of the following values. This property will be null in Common.Status for journal printers or if 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. default: null

Properties
printer/paper Specifies the state of paper supplies as one of the following values. Each individual supply state will be null in Common.Status 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. default: null
printer/paper/upper The state of the upper paper supply. default: null
printer/paper/lower The state of the lower paper supply. default: null
printer/paper/external The state of the external paper supply. default: null
printer/paper/aux The state of the auxiliary paper supply. default: null
printer/paper/aux2 The state of the second auxiliary paper supply. default: null
printer/paper/park The state of the parking station paper supply. default: null
printer/paper/vendorSpecificPaperSupply (example name) The state of the additional vendor specific paper supplies. default: null
printer/toner Specifies the state of the toner or ink supply or the state of the ribbon. The property will be null in Common.Status if the capability is not supported by device, otherwise one of the following: • `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. default: null
printer/ink Specifies the status of the stamping ink in the printer. The property will be null in Common.Status if the capability is not supported by device, otherwise one of the following: • `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. default: null

Properties
printer/lamp Specifies the status of the printer imaging lamp. The property will be null in Common.Status if the capability is not supported by device, otherwise one of the following: • `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. default: null
printer/retractBins An array of bin state objects. If no retain bins are supported, the property will be null. default: null
printer/retractBins/state Specifies the state of the printer retract bin as one of the following. This may be null in Common.StatusChangedEvent if unchanged. • `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. default: null
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. This may be null in Common.StatusChangedEvent if unchanged. Property value constraints: `minimum: 0` default: null
printer/mediaOnStacker The number of media on stacker; applicable only to printers with stacking capability therefore null if not applicable. Property value constraints: `minimum: 0` default: null
printer/paperType Specifies the type of paper loaded as one of the following. Only applicable properties are reported. This may be null in Common.StatusChangedEvent if unchanged. • `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. default: null
printer/paperType/upper The upper paper supply paper type. default: null
printer/paperType/lower The lower paper supply paper type. default: null
printer/paperType/external The external paper supply paper type. default: null

Properties
printer/paperType/aux The auxililliary paper supply paper type. default: null
printer/paperType/aux2 The second auxililliary paper supply paper type. default: null
printer/paperType/park The parking station paper supply paper type. default: null
printer/paperType/exampleProperty1 (example name) The additional vendor specific paper types. default: null
printer/blackMarkMode Specifies the status of the black mark detection and associated functionality. The property is null if 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. default: null
barcodeReader Status information for XFS4IoT services implementing the Barcode Reader interface. This will be null if the Barcode Reader interface is not supported. default: null
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 null if the Biometrics interface is not supported. default: null
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). This property is null if the physical device does not support the ability to report whether or not a subject is on the scanning position. default: null
biometric/capture Indicates whether scanned biometric data has been captured using the Biometric.Read and is currently stored and ready for comparison. This will be set to false when scanned data is cleared using the Biometric.Clear. This may be null in Common.StatusChangedEvent if unchanged. default: null

Properties
biometric/dataPersistence Specifies the current data persistence mode. The data persistence mode controls how biometric data that has been captured using the Biometric.Read will be handled. This property is null if the property persistenceModes is null or both properties persist and clear are false. 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. default: null
biometric/remainingStorage Specifies how much of the reserved storage specified by the capability templateStorage is remaining for the storage of templates in bytes. if null, this property is not supported. Property value constraints: `minimum: 0` default: null
camera Status information for XFS4IoT services implementing the Camera interface. This will be null if the Camera interface is not supported. default: null
camera/media Specifies the state of the recording media of the cameras as one of the following. For a device which stores pictures on a hard disk drive or other general-purpose storage, the relevant property will be null. This property may be null in Common.StatusChangedEvent if unchanged. • `ok` - The media is in a good state. • `high` - The media is almost full (threshold). • `full` - The media is full. • `unknown` - Due to a hardware error or other condition, the state of the media cannot be determined. default: null
camera/media/room Specifies the state of the recording media of the camera that monitors the whole self-service area. default: null
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. default: null
camera/media/exitSlot Specifies the state of the recording media of the camera that monitors the exit slot(s) of the self-service machine. default: null
camera/media/vendorSpecificCameraMedia (example name) Allows vendor specific cameras to be reported. default: null
camera/cameras Specifies the state of the cameras as one of the following. The relevant property will be null if not supported and this property may be null in Common.StatusChangedEvent if unchanged. • `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. default: null

Properties
camera/cameras/room Specifies the state of the camera that monitors the whole self-service area. default: null
camera/cameras/person Specifies the state of the camera that monitors the person standing in front of the self-service machine. default: null
camera/cameras/exitSlot Specifies the state of the camera that monitors the exit slot(s) of the self-service machine. default: null
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 relevant camera's property is 0. Properties may be null in Common.StatusChangedEvent if unchanged. default: null
camera/pictures/room Specifies the number of pictures stored on the recording media of the room camera. Property value constraints: `minimum: 0` default: null
camera/pictures/person Specifies the number of pictures stored on the recording media of the person camera. Property value constraints: `minimum: 0` default: null
camera/pictures/exitSlot Specifies the number of pictures stored on the recording media of the exit slot camera. Property value constraints: `minimum: 0` default: null
camera/pictures/vendorSpecificCameraPictures (example name) Allows vendor specific cameras to be reported. Property value constraints: `minimum: 0` default: null
lights Status information for XFS4IoT services implementing the Lights interface. This will be null if the Lights interface is not supported. default: null
lights/cardReader Card Reader Light. This property is null if not applicable. default: null

Properties
lights/cardReader/position The light position. Can be used for devices which have multiple input and output positions. This may be 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. • `default` - The default position.
lights/cardReader/flashRate The light flash rate. This may be null in Common.StatusChangedEvent if unchanged, otherwise 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). default: null
lights/cardReader/color The light color. This may be null in Common.StatusChangedEvent if unchanged, otherwise 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. default: null
lights/cardReader/direction The light direction, The value can be null if not required. One of the following values: • `entry` - The light is indicating entry. • `exit` - The light is indicating exit. default: null
lights/pinPad Pin Pad Light. This property is null if not applicable. default: null
lights/notesDispenser Notes Dispenser Light. This property is null if not applicable. default: null
lights/coinDispenser Coin Dispenser Light. This property is null if not applicable. default: null
lights/receiptPrinter Receipt Printer Light. This property is null if not applicable. default: null

Properties
lights/passbookPrinter Passbook Printer Light. This property is null if not applicable. default: null
lights/envelopeDepository Envelope Depository Light. This property is null if not applicable. default: null
lights/checkUnit Check Unit Light. This property is null if not applicable. default: null
lights/billAcceptor Bill Acceptor Light. This property is null if not applicable. default: null
lights/envelopeDispenser Envelope Dispenser Light. This property is null if not applicable. default: null
lights/documentPrinter Document Printer Light. This property is null if not applicable. default: null
lights/coinAcceptor Coin Acceptor Light. This property is null if not applicable. default: null
lights/scanner Scanner Light. This property is null if not applicable. default: null
lights/contactless Contactless Reader Light. This property is null if not applicable. default: null
lights/cardReader2 Card Reader 2 Light. This property is null if not applicable. default: null
lights/notesDispenser2 Notes Dispenser 2 Light. This property is null if not applicable. default: null
lights/billAcceptor2 Bill Acceptor 2 Light. This property is null if not applicable. default: null
lights/statusGood Status Indicator light - Good. This property is null if not applicable. default: null
lights/statusWarning Status Indicator light - Warning. This property is null if not applicable. default: null
lights/statusBad Status Indicator light - Bad. This property is null if not applicable. default: null

Properties
lights/statusSupervisor Status Indicator light - Supervisor. This property is null if not applicable. default: null
lights/statusInService Status Indicator light - In Service. This property is null if not applicable. default: null
lights/fasciaLight Fascia Light. This property is null if not applicable. default: null
lights/vendorSpecificLight (example name) Additional vendor specific lights. default: null
auxiliaries Status information for XFS4IoT services implementing the Auxiliaries interface. This will be null if the Auxiliaries interface is not supported. default: null
auxiliaries/operatorSwitch Specifies the state of the Operator switch. • `run` - The switch is in run mode. • `maintenance` - The switch is in maintenance mode. • `supervisor` - The switch is in supervisor mode. This property is null if not applicable. default: null
auxiliaries/tamperSensor Specifies the state of the Tamper sensor. • `off` - There is no indication of a tampering attempt. • `on` - There has been a tampering attempt. This property is null if not applicable. default: null
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: • `off` - There is no indication of a tampering attempt. • `on` - There has been a tampering attempt. This property is null if not applicable. default: null
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: • `off` - The seismic activity has not been high enough to trigger the sensor. • `on` - The seismic or other activity has triggered the sensor. This property is null if not applicable. default: null

Properties
auxiliaries/heatSensor Specifies the state of the Heat Sensor. This sensor is triggered by excessive heat (fire) near the terminal. Specified as one of the following: • `off` - The heat has not been high enough to trigger the sensor. • `on` - The heat has been high enough to trigger the sensor. This property is null if not applicable. default: null
auxiliaries/proximitySensor Specifies the state of the Proximity Sensor. This sensor is triggered by movements around the terminal. Specified as one of the following: • `present` - The sensor is showing that there is someone present at the terminal. • `notPresent` - The sensor can not sense any people around the terminal. This property is null if not applicable. default: null
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: • `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. This property is null if not applicable. default: null
auxiliaries/enhancedAudioSensor Specifies the presence or absence of a consumer′s headphone connected to the Audio Jack. Specified as one of the following: • `present` - There is a headset connected. • `notPresent` - There is no headset connected. This property is null if not applicable. default: null
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: • `off` - The sensor has not been triggered. • `on` - The terminal is about to be rebooted or shutdown. This property is null if not applicable. default: null
auxiliaries/consumerDisplaySensor Specifies the state of the Consumer Display. Specified as one of the following: • `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. This property is null if not applicable. default: null

Properties
auxiliaries/operatorCallButtonSensor Specifies the state of the Operator Call Button as one of the following: • `off` - The Operator Call Button is released (not pressed). • `on` - The Operator Call Button is being pressed. This property is null if not applicable. default: null
auxiliaries/handsetSensor Specifies the state of the Handset, which is a device similar to a telephone receiver. Specified as one of the following: • `onTheHook` - The Handset is on the hook. • `offTheHook` - The Handset is off the hook. This property is null if not applicable. default: null
auxiliaries/headsetMicrophoneSensor Specifies the presence or absence of a consumer′s headset microphone connected to the Microphone Jack. Specified as one of the following: • `present` - There is a headset microphone connected. • `notPresent` - There is no headset microphone connected. This property is null if not applicable. default: null
auxiliaries/fasciaMicrophoneSensor Specifies the state of the fascia microphone as one of the following: • `off` - The Fascia Microphone is turned off. • `on` - The Fascia Microphone is turned on. This property is null if not applicable. default: null
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: • `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. This property is null if not applicable. default: null
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: • `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. This property is null if not applicable. default: null

Properties
auxiliaries/cabinetFrontDoor Specifies the overall state of the Front Cabinet Doors. The front is defined as the side facing the customer/consumer. Cabinet Doors are doors that open up for consumables, and hardware that does not have to be in a secure place. Specified as one of the following: • `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. This property is null if not applicable. default: null
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: • `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. This property is null if not applicable. default: null
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: • `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. This property is null if not applicable. default: null
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: • `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. This property is null if not applicable. default: null
auxiliaries/openClosedIndicator Specifies the state of the Open/Closed Indicator as one of the following: • `closed` - The terminal is closed for a consumer. • `open` - The terminal is open to be used by a consumer. This property is null if not applicable. default: null

Properties
auxiliaries/audio Specifies the state of the Audio Indicator. This property is null if not applicable. default: null
auxiliaries/audio/rate Specifies the state of the Audio Indicator as one of the following values. This may be null in Common.StatusChangedEvent if unchanged. • `on` - Turn on the Audio Indicator. • `off` - Turn off the Audio Indicator. • `continuous` - Turn the Audio Indicator to continuous. This property is null if not applicable. default: null
auxiliaries/audio/signal Specifies the Audio sound as one of the following values. This may be null in Common.StatusChangedEvent if unchanged. • `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. This property is null if not applicable. default: null
auxiliaries/heating Specifies the state of the internal heating as one of the following: • `off` - The internal heating is turned off. • `on` - The internal heating is turned on. This property is null if not applicable. default: null
auxiliaries/consumerDisplayBacklight Specifies the Consumer Display Backlight as one of the following: • `off` - The Consumer Display Backlight is turned off. • `on` - Consumer Display Backlight is turned on. This property is null if not applicable. default: null
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: • `off` - The Signage Display is turned off. • `on` - The Signage Display is turned on. This property is null if not applicable. default: null

Properties

auxiliaries/volume
Specifies the value of the Volume Control. 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. This property is null if not applicable.
Property value constraints:
`minimum: 1`
`maximum: 1000`
default: null

auxiliaries/UPS Specifies the state of the Uninterruptible Power Supply. This property is null if not applicable. Properties contained in this property may be null in Common.StatusChangedEvent if unchanged. default: null
auxiliaries/UPS/low The charge level of the UPS is low. default: null
auxiliaries/UPS/engaged The UPS is engaged. default: null
auxiliaries/UPS/powering The UPS is powering the system. default: null
auxiliaries/UPS/recovered The UPS was engaged when the main power went off. default: null
auxiliaries/audibleAlarm Species the state of the Audible Alarm device as one of the following: • `off` - The Alarm is turned off. • `on` - The Alarm is turned on. This property is null if not applicable. default: null

Properties

auxiliaries/enhancedAudioControl

Specifies the state of the Enhanced Audio Controller. The Enhanced Audio Controller controls how private and public audio are broadcast when the headset is inserted into/removed from the audio jack and when the handset is off-hook/on-hook. In the following, Privacy Device is used to refer to either the headset or handset. The Enhanced Audio Controller state is specified as one of the following:

- `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.

This property is null if not applicable.

default: null

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 following values:

- `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.

This property is null if not applicable.

default: null

auxiliaries/microphoneVolume

Specifies the value of the Microphone Volume Control. 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. This property is null if not applicable.

Property value constraints:

```
minimum: 1
maximum: 1000
```

default: null

vendorMode

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

default: null

vendorMode/device

Specifies the status of the Vendor Mode Service. This property may be null in events if the status did not change, otherwise will be one of the following values:

- `online` - The Vendor Mode service is available.
- `offline` - The Vendor Mode service is not available.

default: null

Properties
vendorMode/service Specifies the service state. This property may be null in events if the state did not change, otherwise will be 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. default: null
vendorApplication Status information for XFS4IoT services implementing the Vendor Application interface. This will be null in Common.Status if the interface is not supported. default: null
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 2.0)
This message does not define any properties.

Completion Message

Payload (version 2.0)	Type	Required
{		
"interfaces": [{	array (object)	✓
"name": "Common",	string	✓
"commands": {	object	✓
"*CardReader.ReadRawData*": {	object	
"versions": ["1.3", "2.1", "3.0"]	array (string)	✓
},		
"*CardReader.Move*": See *interfaces/commands/CardReader.ReadRawData* properties	object	
},		
"events": {	object, null	
"*CardReader.MediaInsertedEvent*": {	object	
"versions": ["1.3", "2.1", "3.0"]	array (string)	✓
},		
"*CardReader.MediaRemovedEvent*": See *interfaces/events/CardReader.MediaInsertedEvent* properties	object	
},		
"maximumRequests": 0	integer	
}],		
"common": {	object	✓
"serviceVersion": "1.3.42",	string	✓
"deviceInformation": [{	array (object)	✓
"modelName": "AcmeModel42",	string, null	
"serialNumber": "1.0.12.05",	string, null	
"revisionNumber": "1.2.3",	string, null	
"modelDescription": "Acme Dispenser Model 3",	string, null	
"firmware": [{	array (object), null	

Payload (version 2.0)	Type	Required
"firmwareName": "Acme Firmware",	string, null	
"firmwareVersion": "1.0.1.2",	string, null	
"hardwareRevision": "4.3.0.5"	string, null	
}],		
"software": [{	array (object), null	
"softwareName": "Acme Software Name",	string, null	
"softwareVersion": "1.3.0.2"	string, null	
}]		
}],		
"powerSaveControl": false,	boolean	
"antiFraudModule": false,	boolean	
"endToEndSecurity": {	object, null	
"required": "always",	string	✓
"hardwareSecurityElement": true,	boolean	✓
"responseSecurityEnabled": "always",	string	
"commands": ["CashDispenser.Dispense"],	array (string)	✓
"commandNonceTimeout": 3600	integer	✓
}		
},		
"cardReader": {	object, null	
"type": "motor",	string	✓
"readTracks": {	object, null	
"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, null	
"track1": false,	boolean	
"track2": false,	boolean	
"track3": false,	boolean	
"frontTrack1": false,	boolean	

Payload (version 2.0)	Type	Required
"track1JIS": false,	boolean	
"track3JIS": false	boolean	
},		
"chipProtocols": {	object, null	
"chipT0": false,	boolean	
"chipT1": false,	boolean	
"chipProtocolNotRequired": false,	boolean	
"chipTypeAPart3": false,	boolean	
"chipTypeAPart4": false,	boolean	
"chipTypeB": false,	boolean	
"chipTypeNFC": false	boolean	
},		
"securityType": "mm",	string, null	
"powerOnOption": "exit",	string, null	
"powerOffOption": "exit",	string, null	
"fluxSensorProgrammable": false,	boolean	
"readWriteAccessFromExit": false,	boolean	
"writeMode": {	object, null	
"loco": false,	boolean	
"hico": false,	boolean	
"auto": false	boolean	
},		
"chipPower": {	object, null	
"cold": false,	boolean	
"warm": false,	boolean	
"off": false	boolean	
},		
"memoryChipProtocols": {	object, null	
"siemens4442": false,	boolean	
"gpm896": false	boolean	
},		
"positions": {	object, null	
"exit": false,	boolean	
"transport": false	boolean	
},		
"cardTakenSensor": false	boolean	
},		
"cashAcceptor": {	object, null	
"type": "tellerBill",	string	✓
"maxCashInItems": 1,	integer	

Payload (version 2.0)	Type	Required
"shutter": false,	boolean	
"shutterControl": false,	boolean	
"intermediateStacker": 0,	integer, null	
"itemsTakenSensor": false,	boolean	
"itemsInsertedSensor": false,	boolean	
"positions": [{	array (object)	✓
"position": "inLeft",	string	✓
"usage": {	object	✓
"in": false,	boolean	
"refuse": false,	boolean	
"rollback": false	boolean	
},		
"shutterControl": true,	boolean	
"itemsTakenSensor": false,	boolean	
"itemsInsertedSensor": false,	boolean	
"retractAreas": {	object, null	
"retract": false,	boolean	
"reject": false,	boolean	
"transport": false,	boolean	
"stacker": false,	boolean	
"billCassettes": false,	boolean	
"cashIn": false	boolean	
},		
"presentControl": false,	boolean	
"preparePresent": false	boolean	
}],		
"retractAreas": {	object, null	
"retract": false,	boolean	
"transport": false,	boolean	
"stacker": false,	boolean	
"reject": false,	boolean	
"billCassette": false,	boolean	
"cashIn": false	boolean	
},		
"retractTransportActions": {	object, null	
"present": false,	boolean	
"retract": false,	boolean	
"reject": false,	boolean	
"billCassette": false,	boolean	

Payload (version 2.0)	Type	Required
"cashIn": See cashAcceptor/retractAreas/cashIn	boolean	
},		
"retractStackerActions": See cashAcceptor/retractTransportActions properties	object, null	
"cashInLimit": {	object, null	
"byTotalItems": false,	boolean	
"byAmount": false	boolean	
},		
"countActions": {	object, null	
"individual": false,	boolean	
"all": false	boolean	
},		
"retainAction": {	object, null	
"counterfeit": false,	boolean	
"suspect": false,	boolean	
"inked": false	boolean	
}		
},		
"cashDispenser": {	object, null	
"type": "tellerBill",	string	✓
"maxDispenseItems": 1,	integer	✓
"shutterControl": false,	boolean	
"retractAreas": {	object, null	
"retract": false,	boolean	
"transport": false,	boolean	
"stacker": false,	boolean	
"reject": false,	boolean	
"itemCassette": false,	boolean	
"cashIn": false	boolean	
},		
"retractTransportActions": {	object, null	
"present": false,	boolean	
"retract": false,	boolean	
"reject": false,	boolean	
"itemCassette": false,	boolean	
"cashIn": false	boolean	
},		
"retractStackerActions": See cashDispenser/retractTransportActions properties	object, null	
"intermediateStacker": false,	boolean	
"itemsTakenSensor": false,	boolean	

Payload (version 2.0)	Type	Required
"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, null	
"fromCashUnit": false,	boolean	
"toCashUnit": false,	boolean	
"toTransport": false,	boolean	
"toStacker": false	boolean	
}		
},		
"cashManagement": {	object, null	
"cashBox": false,	boolean	
"exchangeType": {	object	✓
"byHand": false	boolean	
},		
"itemInfoTypes": {	object, null	
"serialNumber": false,	boolean	
"signature": false,	boolean	
"image": false	boolean	
},		
"classificationList": false,	boolean	
"classifications": {	object	
"unrecognized": true,	boolean	
"counterfeit": false,	boolean	
"suspect": false,	boolean	
"inked": false,	boolean	
"fit": true,	boolean	
"unfit": false	boolean	
}		
},		
"check": {	object, null	
"type": "singleMediaInput",	string	✓
"maxMediaOnStacker": 0,	integer	
"printSize": {	object, null	

Payload (version 2.0)	Type	Required
"rows": 0,	integer	
"cols": 0	integer	
},		
"stamp": false,	boolean	
"rescan": false,	boolean	
"presentControl": false,	boolean	✓
"applicationRefuse": false,	boolean	
"retractLocation": {	object, null	
"storage": false,	boolean	
"transport": false,	boolean	
"stacker": false,	boolean	
"rebuncher": false	boolean	
},		
"resetControl": {	object, null	
"eject": false,	boolean	
"storageUnit": false,	boolean	
"transport": See check/retractLocation/transport,	boolean	
"rebuncher": See check/retractLocation/rebuncher	boolean	
},		
"imageType": {	object, null	
"tif": false,	boolean	
"wmf": false,	boolean	
"bmp": false,	boolean	
"jpg": false	boolean	
},		
"frontImage": {	object, null	
"colorFormat": {	object	✓
"binary": false,	boolean	
"grayScale": false,	boolean	
"full": false	boolean	
},		
"scanColor": {	object	✓
"red": false,	boolean	
"green": false,	boolean	
"blue": false,	boolean	
"yellow": false,	boolean	
"white": false,	boolean	
"infraRed": false,	boolean	
"ultraViolet": false	boolean	
},		

Payload (version 2.0)	Type	Required
"defaultScanColor": "red"	string	✓
},		
"backImage": See check/frontImage properties	object, null	
"codelineFormat": {	object	✓
"cmc7": false,	boolean	
"e13b": false,	boolean	
"ocr": false,	boolean	
"ocra": false,	boolean	
"ocrb": false	boolean	
},		
"dataSource": {	object	✓
"imageFront": false,	boolean	
"imageBack": false,	boolean	
"codeLine": false	boolean	
},		
"insertOrientation": {	object	✓
"codeLineRight": false,	boolean	
"codeLineLeft": false,	boolean	
"codeLineBottom": false,	boolean	
"codeLineTop": false,	boolean	
"faceUp": false,	boolean	
"faceDown": false	boolean	
},		
"positions": {	object	✓
"input": {	object, null	
"itemsTakenSensor": false,	boolean, null	
"itemsInsertedSensor": false,	boolean, null	
"retractAreas": {	object, null	
"retractBin": false,	boolean	
"transport": false,	boolean	
"stacker": false,	boolean	
"rebuncher": false	boolean	
}		
},		
"output": See check/positions/input properties	object, null	
"refused": See check/positions/input properties	object, null	
},		
"imageAfterEndorse": false,	boolean	

Payload (version 2.0)	Type	Required
"returnedItemsProcessing": {	object, null	
"endorse": false,	boolean	
"endorseImage": false	boolean	
},		
"printSizeFront": See check/printSize properties	object, null	
},		
"mixedMedia": {	object, null	
"modes": {	object	✓
"cashAccept": true,	boolean, null	
"checkAccept": true	boolean, null	
},		
"dynamic": false	boolean	
},		
"pinPad": {	object, null	
"pinFormats": {	object	
"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	
"banksys": false,	boolean	
"emv": false,	boolean	
"iso3": false,	boolean	
"ap": false,	boolean	
"iso4": false	boolean	
},		
"presentationAlgorithms": {	object, null	
"presentClear": false	boolean	
},		
"display": {	object, null	
"none": false,	boolean	
"ledThrough": false,	boolean	
"display": false	boolean	
},		

Payload (version 2.0)	Type	Required
"idcConnect": false,	boolean	
"validationAlgorithms": {	object, null	
"des": false,	boolean	
"visa": false	boolean	
},		
"pinCanPersistAfterUse": false,	boolean	
"typeCombined": false,	boolean	
"setPinblockDataRequired": false,	boolean	
"pinBlockAttributes": {	object, null	
"P0": {	object	
"T": {	object	
"E": {	object	
"cryptoMethod": {	object	✓
"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": See *pinPad/pinBlockAttributes/P0/T* properties	object	
}		
}		
},		
"crypto": {	object, null	
"emvHashAlgorithm": {	object, null	
"sha1Digest": false,	boolean	
"sha256Digest": false	boolean	
},		
"cryptoAttributes": {	object, null	
"D0": {	object	
"D": {	object	
"D": {	object	
"cryptoMethod": {	object	✓
"ecb": false,	boolean	
"cbc": false,	boolean	

Payload (version 2.0)	Type	Required
"cfb": false,	boolean	
"ofb": false,	boolean	
"ctr": false,	boolean	
"xts": false,	boolean	
"rsaesPkcs1V15": false,	boolean	
"rsaesOaep": false	boolean	
}		
},		
"E": See crypto/cryptoAttributes/D0/D/D properties	object	
},		
"T": See crypto/cryptoAttributes/D0/D properties	object	
},		
"D1": See crypto/cryptoAttributes/D0 properties	object	
},		
"authenticationAttributes": {	object, null	
"M0": {	object	
"T": {	object	
"G": {	object, null	
"cryptoMethod": {	object, null	
"rsassaPkcs1V15": false,	boolean	
"rsassaPss": false	boolean	
},		
"hashAlgorithm": {	object, null	
"sha1": false,	boolean	
"sha256": false	boolean	
}		
},		
"S": See crypto/authenticationAttributes/M0/T/G properties	object, null	
},		
"R": See crypto/authenticationAttributes/M0/T properties	object	
},		
"S0": See crypto/authenticationAttributes/M0 properties	object	
},		
"verifyAttributes": {	object, null	
"M0": See crypto/authenticationAttributes/M0	object	
"T": {	object	
"V": {	object, null	
"cryptoMethod": See crypto/authenticationAttributes/M0/T/G/cryptoMethod properties	object, null	

Payload (version 2.0)	Type	Required
"hashAlgorithm": See crypto/authenticationAttributes/M0/T/G/hashAlgorithm properties	object, null	
}		
},		
"R": See crypto/verifyAttributes/M0/T properties	object	
},		
"S0": See crypto/authenticationAttributes/M0	object	
"T": See crypto/verifyAttributes/M0/T properties	object	
"R": See crypto/verifyAttributes/M0/T properties	object	
}		
}		
},		
"keyManagement": {	object, null	
"keyNum": 0,	integer	✓
"derivationAlgorithms": {	object, null	
"chipZka": false	boolean	
},		
"keyCheckModes": {	object, null	
"self": false,	boolean	
"zero": false	boolean	
},		
"hsmVendor": "HSM Vendor",	string, null	
"rsaAuthenticationScheme": {	object, null	
"twoPartySig": false,	boolean	
"threePartyCert": false,	boolean	
"threePartyCertTr34": false	boolean	
},		
"rsaSignatureAlgorithm": {	object, null	
"pkcs1V15": false,	boolean	
"pss": false	boolean	
},		
"rsaCryptAlgorithm": {	object, null	
"pkcs1V15": false,	boolean	
"oaep": false	boolean	
},		
"rsaKeyCheckMode": {	object, null	
"sha1": false,	boolean	
"sha256": false	boolean	
},		
"signatureScheme": {	object, null	

Payload (version 2.0)	Type	Required
"randomNumber": false,	boolean	
"exportDeviceId": false,	boolean	
"enhancedRkl": false	boolean	
},		
"emvImportSchemes": {	object, null	
"plainCA": false,	boolean	
"chksumCA": false,	boolean	
"epiCA": false,	boolean	
"issuer": false,	boolean	
"icc": false,	boolean	
"iccPin": false,	boolean	
"pkcsv15CA": false	boolean	
},		
"keyBlockImportFormats": {	object, null	
"A": false,	boolean	
"B": false,	boolean	
"C": false,	boolean	
"D": false	boolean	
},		
"keyImportThroughParts": false,	boolean	
"desKeyLength": {	object, null	
"single": false,	boolean	
"double": false,	boolean	
"triple": false	boolean	
},		
"certificateTypes": {	object, null	
"encKey": false,	boolean	
"verificationKey": false,	boolean	
"hostKey": false	boolean	
},		
"loadCertOptions": {	object, null	
"*certHost*": {	object	
"newHost": false,	boolean	
"replaceHost": false	boolean	
},		
"*caTr34*": See *keyManagement/loadCertOptions/certHost* properties	object	
},		
"crklLoadOptions": {	object, null	
"noRandom": false,	boolean	

Payload (version 2.0)	Type	Required
"noRandomCrl": false,	boolean	
"random": false,	boolean	
"randomCrl": false	boolean	
},		
"symmetricKeyManagementMethods": {	object, null	
"fixedKey": false,	boolean	
"masterKey": false,	boolean	
"tdesDukpt": false	boolean	
},		
"keyAttributes": {	object	
"M0": {	object	
"T": {	object	
"C": {	object, null	
"restrictedKeyUsage": "string"	string, null	
},		
"E": See keyManagement/keyAttributes/M0/T/C properties	object, null	
},		
"R": See keyManagement/keyAttributes/M0/T properties	object	
},		
"K1": See keyManagement/keyAttributes/M0 properties	object	
},		
"decryptAttributes": {	object, null	
"A": {	object	
"decryptMethod": {	object	✓
"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": See keyManagement/decryptAttributes/A properties	object	
},		
"verifyAttributes": {	object, null	
"M0": {	object	
"T": {	object	
"V": {	object	

Payload (version 2.0)	Type	Required
"cryptoMethod": {	object	✓
"kcvNone": false,	boolean	
"kcvSelf": false,	boolean	
"kcvZero": false,	boolean	
"sigNone": false,	boolean	
"rsassaPkcs1V15": false,	boolean	
"rsassaPss": false	boolean	
},		
"hashAlgorithm": {	object	✓
"sha1": false,	boolean	
"sha256": false	boolean	
}		
},		
"*S*": See *keyManagement/verifyAttributes/M0/T/V* properties	object	
},		
"*R*": See *keyManagement/verifyAttributes/M0/T* properties	object	
},		
"*S0*": See *keyManagement/verifyAttributes/M0* properties	object	
}		
},		
"keyboard": {	object, null	
"autoBeep": {	object, null	
"activeAvailable": false,	boolean	
"activeSelectable": false,	boolean	
"inactiveAvailable": false,	boolean	
"inactiveSelectable": false	boolean	
},		
"etsCaps": {	object, null	
"xPos": 0,	integer	
"yPos": 0,	integer	
"xSize": 0,	integer	
"ySize": 0,	integer	
"maximumTouchFrames": 0,	integer	
"maximumTouchKeys": 0,	integer	
"float": {	object, null	
"x": false,	boolean	
"y": false	boolean	
}		
}		

Payload (version 2.0)	Type	Required
},		
"textTerminal": {	object, null	
"type": "fixed",	string	✓
"resolutions": [{	array (object)	✓
"sizeX": 0,	integer	✓
"sizeY": 0	integer	✓
}],		
"keyLock": false,	boolean	✓
"cursor": false,	boolean	✓
"forms": false	boolean	✓
},		
"printer": {	object, null	
"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, null	
"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": See printer/readForm/ocr,	boolean	

Payload (version 2.0)	Type	Required
"micr": See printer/readForm/micr,	boolean	
"msf": See printer/readForm/msf,	boolean	
"barcode": See printer/readForm/barcode,	boolean	
"stamp": false	boolean	
},		
"extents": {	object, null	
"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	

Payload (version 2.0)	Type	Required
"*exampleProperty2*": See *printer/paperSources/exampleProperty1*	boolean	
},		
"mediaTaken": false,	boolean	
"retractBins": 1,	integer	
"maxRetract": [10],	array (integer), null	
"imageType": {	object, null	
"tif": false,	boolean	
"wmf": false,	boolean	
"bmp": false,	boolean	
"jpg": false	boolean	
},		
"frontImageColorFormat": {	object, null	
"binary": false,	boolean	
"grayscale": false,	boolean	
"full": false	boolean	
},		
"backImageColorFormat": {	object, null	
"binary": See printer/frontImageColorFormat/binary,	boolean	
"grayScale": See printer/frontImageColorFormat/grayscale,	boolean	
"full": See printer/frontImageColorFormat/full	boolean	
},		
"imageSource": {	object, null	
"imageFront": false,	boolean	
"imageBack": false	boolean	
},		
"dispensePaper": false,	boolean	
"osPrinter": "example printer",	string, null	
"mediaPresented": false,	boolean	
"autoRetractPeriod": 0,	integer	
"retractToTransport": false,	boolean	
"coercivityType": {	object, null	
"low": false,	boolean	
"high": false,	boolean	
"auto": false	boolean	
},		
"controlPassbook": {	object, null	
"turnForward": false,	boolean	
"turnBackward": false,	boolean	

Payload (version 2.0)	Type	Required
"closeForward": false,	boolean	
"closeBackward": false	boolean	
},		
"printSides": "single"	string, null	
},		
"barcodeReader": {	object, null	
"canFilterSymbologies": false,	boolean	✓
"symbologies": {	object, null	
"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	

Payload (version 2.0)	Type	Required
"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	
"trioptiCode39": 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, null	
"type": {	object	✓
"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	

Payload (version 2.0)	Type	Required
"signature": false	boolean	
},		
"maxCapture": 0,	integer	✓
"templateStorage": 0,	integer	✓
"dataFormats": {	object	✓
"isoFid": false,	boolean	
"isoFmd": false,	boolean	
"ansiFid": false,	boolean	
"ansiFmd": false,	boolean	
"qso": false,	boolean	
"wso": false,	boolean	
"reservedRaw1": false,	boolean	
"reservedTemplate1": false,	boolean	
"reservedRaw2": See biometric/dataFormats/reservedRaw1,	boolean	
"reservedTemplate2": See biometric/dataFormats/reservedTemplate1,	boolean	
"reservedRaw3": See biometric/dataFormats/reservedRaw1,	boolean	
"reservedTemplate3": See biometric/dataFormats/reservedTemplate1	boolean	
},		
"encryptionAlgorithms": {	object, null	
"ecb": false,	boolean	
"cbc": false,	boolean	
"cfb": false,	boolean	
"rsa": false	boolean	
},		
"storage": {	object, null	
"secure": false,	boolean	
"clear": false	boolean	
},		
"persistenceModes": {	object, null	
"persist": false,	boolean	
"clear": false	boolean	
},		
"matchSupported": "storedMatch",	string, null	
"scanModes": {	object	✓
"scan": false,	boolean	
"match": false	boolean	
},		
"compareModes": {	object, null	
"verify": false,	boolean	

Payload (version 2.0)	Type	Required
"identity": false	boolean	
},		
"clearData": {	object, null	
"scannedData": false,	boolean	
"importedData": false,	boolean	
"setMatchedData": false	boolean	
}		
},		
"camera": {	object, null	
"cameras": {	object	✓
"room": false,	boolean	
"person": false,	boolean	
"exitSlot": false,	boolean	
"*vendorSpecificCamera*": false	boolean	
},		
"maxPictures": 0,	integer, null	
"camData": {	object, null	
"autoAdd": false,	boolean	
"manAdd": false	boolean	
},		
"maxDataLength": 0	integer, null	
},		
"lights": {	object, null	
"cardReader": {	object, null	
"flashRate": {	object	✓
"off": false,	boolean	
"slow": false,	boolean	
"medium": false,	boolean	
"quick": false,	boolean	
"continuous": true	boolean	
},		
"color": {	object	✓
"red": false,	boolean	
"green": false,	boolean	
"yellow": false,	boolean	
"blue": false,	boolean	
"cyan": false,	boolean	
"magenta": false,	boolean	
"white": false	boolean	
},		

Payload (version 2.0)	Type	Required
"direction": {	object, null	
"entry": false,	boolean	
"exit": false	boolean	
},		
"position": {	object, null	
"left": false,	boolean	
"right": false,	boolean	
"center": false,	boolean	
"top": false,	boolean	
"bottom": false,	boolean	
"front": false,	boolean	
"rear": false	boolean	
}		
},		
"pinPad": See lights/cardReader properties	object, null	
"notesDispenser": See lights/cardReader properties	object, null	
"coinDispenser": See lights/cardReader properties	object, null	
"receiptPrinter": See lights/cardReader properties	object, null	
"passbookPrinter": See lights/cardReader properties	object, null	
"envelopeDepository": See lights/cardReader properties	object, null	
"checkUnit": See lights/cardReader properties	object, null	
"billAcceptor": See lights/cardReader properties	object, null	
"envelopeDispenser": See lights/cardReader properties	object, null	
"documentPrinter": See lights/cardReader properties	object, null	
"coinAcceptor": See lights/cardReader properties	object, null	
"scanner": See lights/cardReader properties	object, null	
"contactless": See lights/cardReader properties	object, null	
"cardReader2": See lights/cardReader properties	object, null	
"notesDispenser2": See lights/cardReader properties	object, null	
"billAcceptor2": See lights/cardReader properties	object, null	
"statusGood": See lights/cardReader properties	object, null	
"statusWarning": See lights/cardReader properties	object, null	
"statusBad": See lights/cardReader properties	object, null	
"statusSupervisor": See lights/cardReader properties	object, null	
"statusInService": See lights/cardReader properties	object, null	
"fasciaLight": See lights/cardReader properties	object, null	
"*vendorSpecificLight*": See *lights/cardReader* properties	object, null	
},		
"auxiliaries": {	object, null	
"operatorSwitch": {	object, null	

Payload (version 2.0)	Type	Required
"run": false,	boolean	
"maintenance": false,	boolean	
"supervisor": false	boolean	
},		
"tamperSensor": false,	boolean	
"internalTamperSensor": false,	boolean	
"seismicSensor": false,	boolean	
"heatSensor": false,	boolean	
"proximitySensor": false,	boolean	
"ambientLightSensor": false,	boolean	
"enhancedAudioSensor": {	object, null	
"manual": false,	boolean	
"auto": false,	boolean	
"semiAuto": false,	boolean	
"bidirectional": false	boolean	
},		
"bootSwitchSensor": false,	boolean	
"consumerDisplaySensor": false,	boolean	
"operatorCallButtonSensor": false,	boolean	
"handsetSensor": {	object, null	
"manual": false,	boolean	
"auto": false,	boolean	
"semiAuto": false,	boolean	
"microphone": false	boolean	
},		
"headsetMicrophoneSensor": {	object, null	
"manual": false,	boolean	
"auto": false,	boolean	
"semiAuto": false	boolean	
},		
"fasciaMicrophoneSensor": false,	boolean	
"cabinetDoor": {	object, null	
"closed": false,	boolean	
"open": false,	boolean	
"locked": false,	boolean	
"bolted": false,	boolean	
"tampered": false	boolean	
},		
"safeDoor": See auxiliaries/cabinetDoor properties	object, null	
"vandalShield": {	object, null	

Payload (version 2.0)	Type	Required
"closed": false,	boolean	
"open": false,	boolean	
"locked": false,	boolean	
"service": false,	boolean	
"keyboard": false,	boolean	
"tampered": false	boolean	
},		
"frontCabinet": See auxiliaries/cabinetDoor properties	object, null	
"rearCabinet": See auxiliaries/cabinetDoor properties	object, null	
"leftCabinet": See auxiliaries/cabinetDoor properties	object, null	
"rightCabinet": See auxiliaries/cabinetDoor properties	object, null	
"openCloseIndicator": false,	boolean	
"audio": false,	boolean	
"heating": false,	boolean	
"consumerDisplayBacklight": false,	boolean	
"signageDisplay": false,	boolean	
"volume": 1,	integer	
"ups": {	object, null	
"low": false,	boolean	
"engaged": false,	boolean	
"powering": false,	boolean	
"recovered": false	boolean	
},		
"audibleAlarm": false,	boolean	
"enhancedAudioControl": {	object, null	
"headsetDetection": false,	boolean	
"modeControllable": false	boolean	
},		
"enhancedMicrophoneControl": {	object, null	
"headsetDetection": false,	boolean	
"modeControllable": false	boolean	
},		
"microphoneVolume": 1,	integer, null	
"autoStartupMode": {	object, null	
"specific": false,	boolean	
"daily": false,	boolean	
"weekly": false	boolean	
}		
},		
"vendorApplication": {	object, null	

Payload (version 2.0)	Type	Required
"supportedAccessLevels": {	object, null	✓
"basic": false,	boolean	
"intermediate": false,	boolean	
"full": false	boolean	
}		
}		
}		

Properties

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.
- `Biometric` - Biometric interface.
- `Check` - Check interface.
- `MixedMedia` - MixedMedia 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. May be null if no events are supported.
default: null

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 0 if the maximum number of requests is unlimited. Property value constraints: `minimum: 0` default: 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 null if the device model name is unknown. default: null
common/deviceInformation/serialNumber Specifies the unique serial number of the device. The property is null if the serial number is unknown. default: null
common/deviceInformation/revisionNumber Specifies the device revision number. The property is null if the device revision number is unknown. default: null
common/deviceInformation/modelDescription Contains a description of the device. The property is null if the model description is unknown. default: null
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 null. default: null
common/deviceInformation/firmware/firmwareName Specifies the firmware name. The property is null if the firmware name is unknown. default: null
common/deviceInformation/firmware/firmwareVersion Specifies the firmware version. The property is null if the firmware version is unknown. default: null
common/deviceInformation/firmware/hardwareRevision Specifies the hardware revision. The property is null if the hardware revision is unknown. default: null

Properties
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 null. default: null
common/deviceInformation/software/softwareName Specifies the software name. The property is null if the software name is unknown. default: null
common/deviceInformation/software/softwareVersion Specifies the software version. The property is null if the software version is unknown. default: null
common/powerSaveControl Specifies whether power saving control is available. default: false
common/antiFraudModule Specifies whether the anti-fraud module is available. default: false
common/endToEndSecurity If 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 is null then end to end security is not supported by this service. default: null
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.
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. This property is null in Common.Status if the device is incapable of returning a response token, otherwise one of the following values: • `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.

Properties

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]*$
```

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 Common.ClearCommandNonce command.

Property value constraints:

```
minimum: 0
```

cardReader

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

default: null

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. May be null if not applicable.

default: null

Properties
cardReader/readTracks/track1 The card reader can access track 1. default: false
cardReader/readTracks/track2 The card reader can access track 2. default: false
cardReader/readTracks/track3 The card reader can access track 3. default: false
cardReader/readTracks/watermark The card reader can access the Swedish watermark track. default: false
cardReader/readTracks/frontTrack1 The card reader can access front track 1. default: false
cardReader/readTracks/frontImage The card reader can read the front image of the card. default: false
cardReader/readTracks/backImage The card reader can read the back image of the card. default: false
cardReader/readTracks/track1JIS The card reader can access JIS I track 1. default: false
cardReader/readTracks/track3JIS The card reader can access JIS I track 3. default: false
cardReader/readTracks/ddi The card reader can provide dynamic digital identification of the magnetic strip. default: false
cardReader/writeTracks Specifies the tracks that can be written by the card reader. May be null if no tracks can be written. default: null
cardReader/writeTracks/track1 The card reader can write on track 1. default: false
cardReader/writeTracks/track2 The card reader can write on track 2. default: false
cardReader/writeTracks/track3 The card reader can write on track 3. default: false
cardReader/writeTracks/frontTrack1 The card reader can write on front track 1. default: false

Properties
cardReader/writeTracks/track1JIS The card reader can write on JIS I track 1. default: false
cardReader/writeTracks/track3JIS The card reader can write on JIS I track 3. default: false
cardReader/chipProtocols Specifies the chip card protocols that are supported by the card reader. May be null if none are supported. default: null
cardReader/chipProtocols/chipT0 The card reader can handle the T=0 protocol. default: false
cardReader/chipProtocols/chipT1 The card reader can handle the T=1 protocol. default: false
cardReader/chipProtocols/chipProtocolNotRequired The carder is capable of communicating with the chip without requiring the application to specify any protocol. default: false
cardReader/chipProtocols/chipTypeAPart3 The card reader can handle the ISO 14443 (Part3) Type A contactless chip card protocol. default: false
cardReader/chipProtocols/chipTypeAPart4 The card reader can handle the ISO 14443 (Part4) Type A contactless chip card protocol. default: false
cardReader/chipProtocols/chipTypeB The card reader can handle the ISO 14443 Type B contactless chip card protocol. default: false
cardReader/chipProtocols/chipTypeNFC The card reader can handle the ISO 18092 (106/212/424kbps) contactless chip card protocol. default: false
cardReader/securityType Specifies the type of security module as one of the following or null if the device has no security module. • `mm` - The security module is a MMBox. • `cim86` - The security module is a CIM86. default: null
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). May be null if 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. default: null

Properties
cardReader/powerOffOption Specifies the power-off capabilities of the device hardware. See powerOnOption. default: null
cardReader/fluxSensorProgrammable Specifies whether the Flux Sensor on the card unit is programmable. default: false
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. default: false
cardReader/writeMode The write capabilities, with respect to whether the device can write low coercivity (loco) and/or high coercivity (hico) magnetic stripes. May be null if not applicable. default: null
cardReader/writeMode/loco Supports writing of loco magnetic stripes. default: false
cardReader/writeMode/hico Supports writing of hico magnetic stripes. default: false
cardReader/writeMode/auto The Service is capable of automatically determining whether loco or hico magnetic stripes should be written. default: false
cardReader/chipPower The chip power management capabilities (in relation to the user or permanent chip controlled by the Service. May be null if not applicable. default: null
cardReader/chipPower/cold The card reader can power on the chip and reset it (Cold Reset). default: false
cardReader/chipPower/warm The card reader can reset the chip (Warm Reset). default: false
cardReader/chipPower/off The card reader can power off the chip. default: false
cardReader/memoryChipProtocols The memory card protocols that are supported. May be null if not applicable. default: null
cardReader/memoryChipProtocols/siemens4442 The device supports the Siemens 4442 Card Protocol (also supported by the Gemplus GPM2K card). default: false
cardReader/memoryChipProtocols/gpm896 The device supports the Gemplus GPM 896 Card Protocol. default: false

Properties
cardReader/positions Specifies the target positions that are supported for the CardReader.Move command. This is independent of the storage units. May be null if not applicable. default: null
cardReader/positions/exit The device can move a card to the exit position. In this position, the card is accessible to the user. default: false
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. default: false
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. default: false
cashAcceptor Capability information for XFS4IoT services implementing the CashAcceptor interface. This will be null if the CashAcceptor interface is not supported. default: null
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` default: 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. default: false
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 property applies to all shutters and all positions. default: false

Properties
cashAcceptor/intermediateStacker Specifies the number of items the intermediate stacker for cash-in can hold. Will be null if there is no intermediate stacker for cash-in available. Property value constraints: `minimum: 0` default: null
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. default: false
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. default: false
cashAcceptor/positions Array of position capabilities for all positions configured in this service.
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/usage Indicates if a position is used to input, reject or rollback.
cashAcceptor/positions/usage/in It is an input position. default: false
cashAcceptor/positions/usage/refuse It is an output position used to refuse items. default: false
cashAcceptor/positions/usage/rollback It is an output position used to rollback items. default: false

Properties
cashAcceptor/positions/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* property is always true if the described position has no shutter. default: true
cashAcceptor/positions/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 property relates to output and refused positions. default: false
cashAcceptor/positions/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 property relates to all input positions. default: false
cashAcceptor/positions/retractAreas Specifies the areas to which items may be retracted from this position. This is not reported if the device cannot retract. default: null
cashAcceptor/positions/retractAreas/retract Items may be retracted to a retract storage unit. default: false
cashAcceptor/positions/retractAreas/reject Items may be retracted to a reject storage unit. default: false
cashAcceptor/positions/retractAreas/transport Items may be retracted to the transport. default: false
cashAcceptor/positions/retractAreas/stacker Items may be retracted to the intermediate stacker. default: false
cashAcceptor/positions/retractAreas/billCassettes Items may be retracted to item cassettes, i.e. cash-in and recycle storage units. default: false
cashAcceptor/positions/retractAreas/cashIn Items may be retracted to a cash-in storage unit. default: false

Properties
cashAcceptor/positions/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. default: false
cashAcceptor/positions/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. default: false
cashAcceptor/retractAreas Specifies the area to which items may be retracted. If the device does not have a retract capability this will be null. default: null
cashAcceptor/retractAreas/retract The items may be retracted to a retract storage unit. default: false
cashAcceptor/retractAreas/transport The items may be retracted to the transport. default: false
cashAcceptor/retractAreas/stacker The items may be retracted to the intermediate stacker. default: false
cashAcceptor/retractAreas/reject The items may be retracted to a reject storage unit. default: false
cashAcceptor/retractAreas/billCassette The items may be retracted to cash-in and recycle storage units. default: false
cashAcceptor/retractAreas/cashIn Items may be retracted to cash-in storage units. default: false
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 this will be null. default: null
cashAcceptor/retractTransportActions/present The items may be presented. default: false
cashAcceptor/retractTransportActions/retract The items may be moved to a retract storage unit. default: false

Properties
cashAcceptor/retractTransportActions/reject The items may be moved to a reject storage unit. default: false
cashAcceptor/retractTransportActions/billCassette The items may be moved to the cash-in and recycle storage units. default: false
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 this will be null. default: null
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 null. default: null
cashAcceptor/cashInLimit/byTotalItems The number of successfully processed cash-in items can be limited by specifying the total number of items. default: false
cashAcceptor/cashInLimit/byAmount The number of successfully processed cash-in items can be limited by specifying the maximum amount of a specific currency. default: false
cashAcceptor/countActions Specifies the count action supported by the CashAcceptor.CashUnitCount command. If the device does not support counting then this property is null. default: null
cashAcceptor/countActions/individual The counting of individual storage units is supported. default: false
cashAcceptor/countActions/all The counting of all storage units is supported. default: false
cashAcceptor/retainAction If counterfeit, inked or suspect items are supported by the Service (see classifications), this specifies whether such items are retained by the device if detected during a cash-in transaction. See acceptor for details of the impact on offering cash-in transactions if unable to retain items due to storage unit status. This applies regardless of whether their specific note type is configured to not be accepted by CashAcceptor.ConfigureNoteTypes. This property may be null if none of these note classifications are supported. default: null
cashAcceptor/retainAction/counterfeit Items classified as counterfeit are retained during a cash-in transaction. default: false
cashAcceptor/retainAction/suspect Items classified as suspect are retained during a cash-in transaction. default: false
cashAcceptor/retainAction/inked Items classified as inked are retained during a cash-in transaction. default: false

Properties
cashDispenser Capability information for XFS4IoT services implementing the CashDispenser interface. This will be null if the CashDispenser interface is not supported. default: null
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. default: false
cashDispenser/retractAreas Specifies the area to which items may be retracted. If the device does not have a retract capability this will be null. default: null
cashDispenser/retractAreas/retract The items may be retracted to a retract storage unit. default: false
cashDispenser/retractAreas/transport The items may be retracted to the transport. default: false
cashDispenser/retractAreas/stacker The items may be retracted to the intermediate stacker. default: false
cashDispenser/retractAreas/reject The items may be retracted to a reject storage unit. default: false
cashDispenser/retractAreas/itemCassette The items may be retracted to storage units which would be used during a Cash In transaction including recycling storage units. default: false
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. default: false
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 this will be null. default: null

Properties
cashDispenser/retractTransportActions/present The items may be presented. default: false
cashDispenser/retractTransportActions/retract The items may be moved to a retract storage unit. default: false
cashDispenser/retractTransportActions/reject The items may be moved to a reject storage unit. default: false
cashDispenser/retractTransportActions/itemCassette The items may be moved to storage units which would be used during a Cash In transaction including recycling storage units. default: false
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. default: false
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 this will be null. default: null
cashDispenser/intermediateStacker Specifies whether the Dispenser supports stacking items to an intermediate position before the items are moved to the exit position. default: false
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. default: false
cashDispenser/positions Specifies the Dispenser output positions which are available.
cashDispenser/positions/left The Dispenser has a left output position. default: false
cashDispenser/positions/right The Dispenser has a right output position. default: false
cashDispenser/positions/center The Dispenser has a center output position. default: false
cashDispenser/positions/top The Dispenser has a top output position. default: false

Properties
cashDispenser/positions/bottom The Dispenser has a bottom output position. default: false
cashDispenser/positions/front The Dispenser has a front output position. default: false
cashDispenser/positions/rear The Dispenser has a rear output position. default: false
cashDispenser/moveItems Specifies the Dispenser move item options which are available. If not applicable, this property is null. default: null
cashDispenser/moveItems/fromCashUnit The Dispenser can dispense items from the storage units to the intermediate stacker while there are items on the transport. default: false
cashDispenser/moveItems/toCashUnit The Dispenser can retract items to the storage units while there are items on the intermediate stacker. default: false
cashDispenser/moveItems/toTransport The Dispenser can retract items to the transport while there are items on the intermediate stacker. default: false
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. default: false
cashManagement Capability information for XFS4IoT services implementing the CashManagement interface. This will be null if the CashManagement interface is not supported. default: null
cashManagement/cashBox This property is only applicable to teller type devices. It specifies whether or not tellers have been assigned a cash box. default: false
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. default: false
cashManagement/itemInfoTypes Specifies the types of information that can be retrieved through the CashManagement.GetItemInfo command. This property is null if not supported. default: null
cashManagement/itemInfoTypes/serialNumber Serial Number of the item. default: false

Properties
cashManagement/itemInfoTypes/signature Signature of the item. default: false
cashManagement/itemInfoTypes/image Image of the item. default: false
cashManagement/classificationList Specifies whether the Service has the capability to maintain a classification list of serial numbers as well as supporting the associated operations. default: false
cashManagement/classifications Specifies the classifications supported - see Note Classification.
cashManagement/classifications/unrecognized Items can be classified as unrecognized. default: true
cashManagement/classifications/counterfeit Items can be recognized as counterfeit. default: false
cashManagement/classifications/suspect Items can be suspected as counterfeit. default: false
cashManagement/classifications/inked Ink-stained items are recognized. default: false
cashManagement/classifications/fit Genuine items fit for recycling are recognized. default: true
cashManagement/classifications/unfit Genuine items not fit for recycling are recognized. default: false
check Capability information for XFS4IoT services implementing the Check interface. This will be null if the Check interface is not supported. default: null
check/type Specifies the type of the physical device. The following values are possible: • `singleMediaInput` - Device accepts a single media item from the customer. • `bunchMediaInput` - Device accepts a bunch of media items from the customer.
check/maxMediaOnStacker Specifies the maximum number of media items that the stacker can hold (zero if the device does not have a stacker). If the device has a bunch media input capability and the stacker is not present or has a capacity of one then the application must process each item inserted sequentially as described in section Multi-Feed Devices without a Stacker. Property value constraints: `minimum: 0` default: 0

Properties
check/printSize Specifies the maximum print capabilities on the back side of the check, null if device has no back printing capabilities. If the media item is inserted in one of the orientations specified in *insertOrientation*, the device will print on the back side of the media. If the media item is inserted in a different orientation to those specified in *insertOrientation* then printing may occur on the front side, upside down or both. default: null
check/printSize/rows Specifies the maximum number of rows of text that the device can print on the media item. This value is 1 for single line printers. Property value constraints: `minimum: 0` default: 0
check/printSize/cols Specifies the maximum number of characters that can be printed on a row. Property value constraints: `minimum: 0` default: 0
check/stamp Specifies whether the device has stamping capabilities. If the media item is inserted in one of the orientations specified in *insertOrientation*, the device will stamp on the front side of the media. If the media item is inserted in a different orientation to those specified in *insertOrientation* then stamping may occur on the back, upside down or both. default: false
check/rescan Specifies whether the device has the capability to either physically rescan media items after they have been inserted into the device or is able to generate any image supported by the device during the ReadImage command (regardless of the images requested during the MediaIn command). If true then the item can be rescanned or the images can be generated using the parameters passed in the *ReadImage* command. If false then all images required (various color, file format, bit depth) must be gathered during execution of the *MediaIn* command. default: false
check/presentControl Specifies how the presenting of media items is controlled during the MediaInEnd and MediaInRollback commands. If true the presenting is controlled implicitly by the Service. If false the presenting must be controlled explicitly by the application using the PresentMedia command. This applies to all positions.
check/applicationRefuse Specifies if the Device supports the MediaIn command mode where the application decides to accept or refuse each media item that has successfully been accepted by the device. If true then the Service supports this mode. If false then the Service does not support this mode (or the device does not have a stacker). default: false
check/retractLocation Specifies the locations to which the media can be retracted using the Check.RetractMedia command, as a combination of these properties. May be null if not supported: Property value constraints: `minProperties: 1` default: null
check/retractLocation/storage Retract the media to a storage unit. default: false

Properties
check/retractLocation/transport Retract the media to the transport. default: false
check/retractLocation/stacker Retract the media to the stacker. default: false
check/retractLocation/rebuncher Retract the media to the re-buncher. default: false
check/resetControl Specifies the manner in which the media can be handled on Reset, as a combination of these properties. May be null if the command is not supported: Property value constraints: `minProperties: 1` default: null
check/resetControl/eject Eject the media. default: false
check/resetControl/storageUnit Retract the media to retract storage unit. default: false
check/imageType Specifies the image format supported by this device, as a combination of these properties. May be null if not supported. Property value constraints: `minProperties: 1` default: null
check/imageType/tif The device can return scanned images in TIFF 6.0 format. default: false
check/imageType/wmf The device can return scanned images in WMF (Windows Metafile) format. default: false
check/imageType/bmp The device can return scanned images in windows BMP format. default: false
check/imageType/jpg The device can return scanned images in JPG format. default: false
check/frontImage Specifies the capabilities of the front image supported by this device. May be null if front images are not supported. default: null
check/frontImage/colorFormat Specifies the image color formats supported by this device, as a combination of these properties:

Properties
check/frontImage/colorFormat/binary The device can return scanned images in binary. default: false
check/frontImage/colorFormat/grayScale The device can return scanned images in gray scale. default: false
check/frontImage/colorFormat/full The device can return scanned images in full color. default: false
check/frontImage/scanColor Specifies the image scan colors supported by this device and individually controllable by the application. Scan colors are used to enhance the scanning results on colored scan media. This value is specified as a combination of these properties:
check/frontImage/scanColor/red The device can return images scanned with red light. default: false
check/frontImage/scanColor/green The device can return images scanned with green light. default: false
check/frontImage/scanColor/blue The device can return images scanned with blue light. default: false
check/frontImage/scanColor/yellow The device can return images scanned with yellow light. default: false
check/frontImage/scanColor/white The device can return images scanned with white light. default: false
check/frontImage/scanColor/infraRed The device can return images scanned with infrared light. default: false
check/frontImage/scanColor/ultraViolet The device can return images scanned with ultraviolet light. default: false
check/frontImage/defaultScanColor Specifies the default image color format used by this device (i.e. when not explicitly set). The following values are possible: • `red` - The default color is red light. • `green` - The default color is green light. • `blue` - The default color is blue light. • `yellow` - The default color is yellow light. • `white` - The default color is white light. • `infraRed` - The default color is infrared light. • `ultraViolet` - The default color is ultraviolet light.
check/backImage Specifies the capabilities of the back image supported by this device. May be null if back images are not supported. default: null

Properties
check/codelineFormat Specifies the code line formats supported by this device, as a combination of these properties: Property value constraints: `minProperties: 1`
check/codelineFormat/cmc7 The device can read MICR CMC7 [Ref. check-4] code lines. default: false
check/codelineFormat/e13b The device can read MICR E13B [Ref. check-3] code lines. default: false
check/codelineFormat/ocr The device can read code lines using Optical Character Recognition. The default or pre-configured OCR font will be used. default: false
check/codelineFormat/ocra The device can read code lines using Optical Character Recognition font A. The ASCII codes will conform to [Ref. check-1]. default: false
check/codelineFormat/ocrb The device can read code lines using Optical Character Recognition font B. The ASCII codes will conform to [Ref. check-2]. default: false
check/dataSource Specifies the reading/imaging capabilities supported by this device, as a combination of these properties: Property value constraints: `minProperties: 1`
check/dataSource/imageFront The device can scan the front image of the document. default: false
check/dataSource/imageBack The device can scan the back image of the document. default: false
check/dataSource/codeLine The device can recognize the code line. default: false
check/insertOrientation Specifies the media item insertion orientations supported by the Device such that hardware features such as MICR reading, endorsing and stamping will be aligned with the correct edges and sides of the media item. Devices may still return code lines and images even if one of these orientations is not used during media insertion. If the media items are inserted in one of the orientations defined in this capability then any printing or stamping will be on the correct side of the media item. If the media is inserted in a different orientation then any printing or stamping may be on the wrong side, upside down or both. This value is reported based on the customer's perspective. This value is a combination of these properties: Property value constraints: `minProperties: 1`
check/insertOrientation/codeLineRight The media item should be inserted short edge first with the code line to the right. default: false

Properties
check/insertOrientation/codeLineLeft The media item should be inserted short edge first with the code line to the left. default: false
check/insertOrientation/codeLineBottom The media item should be inserted long edge first with the code line to the bottom. default: false
check/insertOrientation/codeLineTop The media item should be inserted long edge first with the code line to the top. default: false
check/insertOrientation/faceUp The media item should be inserted with the front of the media item facing up. default: false
check/insertOrientation/faceDown The media item should be inserted with the front of the media item facing down. default: false
check/positions Specifies the capabilities of up to three logical position types. Property value constraints: `minProperties: 1`
check/positions/input Structure that specifies the capabilities of the input position. default: null
check/positions/input/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 MediaTakenEvent. If false this event is not generated. This relates to output and refused positions, so will be null for input positions. default: null
check/positions/input/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 MediaInsertedEvent. If false this event is not generated. This relates to all input positions, so will always be null for input positions. default: null
check/positions/input/retractAreas Specifies the areas to which items may be retracted from this position. May be null if items can not be retracted from this position. Property value constraints: `minProperties: 1` default: null
check/positions/input/retractAreas/retractBin Can retract items in this position to a retract storage unit. default: false
check/positions/input/retractAreas/transport Can retract items in this position to the transport. default: false
check/positions/input/retractAreas/stacker Can retract items in this position to the stacker. default: false

Properties
check/positions/input/retractAreas/rebuncher Can retract items in this position to the re-buncher. default: false
check/positions/output Structure that specifies the capabilities of the output position. default: null
check/positions/refused Structure that specifies the capabilities of the refused position. default: null
check/imageAfterEndorse Specifies whether the device can generate an image after text is printed on the media item. If true then the generation of the image can be specified using the SetMediaParameters command. If false, this functionality is not available. This capability applies to media items whose destination is a storage unit; the *returnedItemsProcessing* capability indicates whether this functionality is supported for media items that are to be returned to the customer. default: false
check/returnedItemsProcessing Specifies the processing that this device supports for media items that are identified to be returned to the customer using the SetMediaParameters command, as a combination of these properties or null if none apply: Property value constraints: `minProperties: 1` default: null
check/returnedItemsProcessing/endorse This device can endorse a media item to be returned to the customer; the endorsement is specified using the SetMediaParameters command. default: false
check/returnedItemsProcessing/endorseImage This device can generate an image of a media item to be returned to the customer after it has been endorsed; the request for the image is specified using the SetMediaParameters command. default: false
check/printSizeFront Reports the printing capabilities of the device on the front side of the check, null if device has no front printing capabilities. If the media item is inserted in one of the orientations specified in *insertOrientation*, the device will print on the front side of the media. If the media item is inserted in a different orientation to those specified in *insertOrientation* then printing may occur on the back side, upside down or both. default: null
mixedMedia Capability information for XFS4IoT services implementing the MixedMedia interface. This will be null if the MixedMedia interface is not supported. default: null
mixedMedia/modes Specifies the transaction modes supported by the Service. Property value constraints: `minProperties: 1`
mixedMedia/modes/cashAccept Specifies whether transactions can accept cash. This property may be null if no change required or its state has not changed in Common.StatusChangedEvent. default: null

Properties
mixedMedia/modes/checkAccept Specifies whether transactions can accept checks. This property may be null if no change required or its state has not changed in Common.StatusChangedEvent. default: null
mixedMedia/dynamic Specifies whether the mode can be modified during a transaction. default: false
pinPad Capability information for XFS4IoT services implementing the PinPad interface. This will be null if the PinPad interface is not supported. default: null
pinPad/pinFormats Supported PIN format.
pinPad/pinFormats/ibm3624 PIN left justified, filled with padding characters, PIN length 4-16 digits. The padding character is a hexadecimal digit in the range 0x00 to 0x0F. default: false
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). default: false
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). default: false
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). default: false
pinPad/pinFormats/eci2 PIN left justified, filled with padding characters, PIN only 4 digits. default: false
pinPad/pinFormats/eci3 PIN is preceded by the length (digit), PIN length 4-6 digits, the padding character can range from 0x0 through 0xF". default: false
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). default: false
pinPad/pinFormats/diebold PIN is padded with the padding character and may be not encrypted, single encrypted or double encrypted. default: false

Properties
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. default: false
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. default: false
pinPad/pinFormats/banksys PIN is encrypted and formatted according to the Banksys PIN block specifications. default: false
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. default: false
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. default: false
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. default: false
pinPad/pinFormats/iso4 PIN is formatted according to ISO 9564-1: 2017 Format-4 (uses AES Encryption). default: false
pinPad/presentationAlgorithms Supported presentation algorithms. This property is null if not supported. default: null
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. default: false
pinPad/display Specifies the type of the display used in the PIN pad module. This property is null if not supported. default: null
pinPad/display/none No display unit. default: false
pinPad/display/ledThrough Lights next to text guide user. default: false
pinPad/display/display A real display is available (this doesn't apply for self-service). default: false

Properties
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. default: false
pinPad/validationAlgorithms Specifies the algorithms for PIN validation supported by the service. This property is null if not supported. default: null
pinPad/validationAlgorithms/des DES algorithm. default: false
pinPad/validationAlgorithms/visa Visa algorithm. default: false
pinPad/pinCanPersistAfterUse Specifies whether the device can retain the PIN after a PIN processing command. default: false
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. default: false
pinPad/setPinblockDataRequired Specifies whether the command PinPad.SetPinblockData must be called before the PIN is entered via Keyboard.GetPin and retrieved via PinPad.GetPinblock. default: false
pinPad/pinBlockAttributes Attributes supported by the PinPad.GetPinblock command to generate encrypted PIN block. default: null

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$`
Property value constraints:
`minProperties: 1`

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]$`
Property value constraints:
`minProperties: 1`

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. default: false
pinPad/pinBlockAttributes/P0/T/E/cryptoMethod/cbc The CBC encryption method. default: false
pinPad/pinBlockAttributes/P0/T/E/cryptoMethod/cfb The CFB encryption method. default: false
pinPad/pinBlockAttributes/P0/T/E/cryptoMethod/ofb The OFB encryption method. default: false
pinPad/pinBlockAttributes/P0/T/E/cryptoMethod/ctr The CTR method defined in NIST SP800-38A (See [Ref. pinpad-7]). default: false
pinPad/pinBlockAttributes/P0/T/E/cryptoMethod/xts The XTS method defined in NIST SP800-38E (See [Ref. pinpad-8]). default: false
pinPad/pinBlockAttributes/P0/T/E/cryptoMethod/rsaesPkcs1V15 The RSAES_PKCS1-v1.5 algorithm. default: false
pinPad/pinBlockAttributes/P0/T/E/cryptoMethod/rsaesOaep The RSAES OAEP algorithm. default: false
crypto Capability information for XFS4IoT services implementing the Crypto interface. This will be null if the Crypto interface is not supported. default: null

Properties
crypto/emvHashAlgorithm Specifies which hash algorithm is supported for the calculation of the HASH. This property is null if not supported. default: null
crypto/emvHashAlgorithm/sha1Digest The SHA 1 digest algorithm is supported by the Crypto.Digest command. default: false
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. default: false
crypto/cryptoAttributes Attributes supported by the Crypto.CryptoData command to encrypt or decrypt data. default: null

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]$`
Property value constraints:
`minProperties: 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]$`
Property value constraints:
`minProperties: 1`

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]$`

Properties
crypto/cryptoAttributes/D0/D/D/cryptoMethod Specifies the cryptographic method supported by the Crypto.CryptoData command. If the key usage is symmetric encryption methods 'D0'), 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 key usage is asymmetric encryption methods 'D1', 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. default: false
crypto/cryptoAttributes/D0/D/D/cryptoMethod/cbc The CBC encryption method. default: false
crypto/cryptoAttributes/D0/D/D/cryptoMethod/cfb The CFB encryption method. default: false
crypto/cryptoAttributes/D0/D/D/cryptoMethod/ofb The OFB encryption method. default: false
crypto/cryptoAttributes/D0/D/D/cryptoMethod/ctr The CTR method defined in NIST SP800-38A (See [Ref. crypto-4]) default: false
crypto/cryptoAttributes/D0/D/D/cryptoMethod/xts The XTS method defined in NIST SP800-38E. (See [Ref. crypto-5]) default: false
crypto/cryptoAttributes/D0/D/D/cryptoMethod/rsaesPkcs1V15 The RSAES_PKCS1-v1.5 algorithm. default: false
crypto/cryptoAttributes/D0/D/D/cryptoMethod/rsaesOaep The RSAES OAEP algorithm. default: false
crypto/authenticationAttributes Attributes supported by the Crypto.GenerateAuthentication command to generate authentication data. default: null

Properties

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]$|^S[0-2]$
```

Property value constraints:

```
minProperties: 1
```

crypto/authenticationAttributes/M0/T (example name)

Specifies the encryption algorithms supported by the Crypto.GenerateAuthentication command. The following value is possible:

- `A` - AES.
- `D` - DEA.
- `R` - RSA.
- `T` - Triple DEA (also referred to as TDEA).

Property name constraints:

```
pattern: ^[ADRT]$
```

Property value constraints:

```
minProperties: 1
```

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]$
```

default: null

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'), this property should be null.

Property value constraints:

```
minProperties: 1
```

default: null

crypto/authenticationAttributes/M0/T/G/cryptoMethod/rsassaPkcs1V15

The RSASSA-PKCS1-v1.5 algorithm.

default: false

Properties
crypto/authenticationAttributes/M0/T/G/cryptoMethod/rsassaPss The RSASSA-PSS algorithm. default: false
crypto/authenticationAttributes/M0/T/G/hashAlgorithm Specifies the hash algorithm supported. If the *key* usage is one of the MAC usages (e.g. 'M0'), this property should be null. Property value constraints: `minProperties: 1` default: null
crypto/authenticationAttributes/M0/T/G/hashAlgorithm/sha1 The SHA 1 digest algorithm. default: false
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]. default: false
crypto/verifyAttributes Attributes supported by the Crypto.VerifyAuthentication command to verify authentication data. default: null

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]$`
Property value constraints:
`minProperties: 1`

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$` default: null
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'), this property should be null. Property value constraints: `minProperties: 1` default: null
crypto/verifyAttributes/M0/T/V/hashAlgorithm Specifies the hash algorithm supported. If the key usage is one of the MAC usages (e.g. 'M0'), this property should be null. Property value constraints: `minProperties: 1` default: null

Properties
keyManagement Capability information for XFS4IoT services implementing the KeyManagement interface. This will be null if the KeyManagement interface is not supported. default: null
keyManagement/keyNum Number of the keys which can be stored in the encryption/decryption module. Property value constraints: `minimum: 0`
keyManagement/derivationAlgorithms Supported derivation algorithms. This property is null if not supported. default: null
keyManagement/derivationAlgorithms/chipZka Algorithm for the derivation of a chip card individual key as described by the German ZKA. default: false
keyManagement/keyCheckModes Specifies the key check modes that are supported to check the correctness of an imported key value. The modes available for each key may depend on security requirements of the algorithm. The algorithm (i.e. DES, 3DES, AES) and use is determined by the algorithm of the key being checked and security requirements. This property is null if not supported. default: null
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. default: false
keyManagement/keyCheckModes/zero The key check value is created by encrypting a zero value with the key. default: false
keyManagement/hsmVendor Identifies the Hardware Security Module (HSM) Vendor. This should be null if not supported or the HSM vendor is unknown. default: null
keyManagement/rsaAuthenticationScheme Specifies the types of Remote Key Loading/Authentication that are supported. This property is null if not supported. default: null
keyManagement/rsaAuthenticationScheme/twoPartySig Two-party Signature based authentication. default: false
keyManagement/rsaAuthenticationScheme/threePartyCert Three-party Certificate based authentication. default: false
keyManagement/rsaAuthenticationScheme/threePartyCertTr34 Three-party Certificate based authentication described by X9 TR34-2019 [Ref. keymanagement-9]. default: false
keyManagement/rsaSignatureAlgorithm Specifies the types of RSA Signature Algorithm that are supported. default: null

Properties
keyManagement/rsaSignatureAlgorithm/pkcs1V15 pkcs1V15 Signatures supported. default: false
keyManagement/rsaSignatureAlgorithm/pss pss Signatures supported. default: false
keyManagement/rsaCryptAlgorithm Specifies the types of RSA Encipherment Algorithm that are supported. This property is null if not supported. default: null
keyManagement/rsaCryptAlgorithm/pkcs1V15 pkcs1V15 algorithm supported. default: false
keyManagement/rsaCryptAlgorithm/oaep oaep algorithm supported. default: false
keyManagement/rsaKeyCheckMode Specifies which hash algorithms used to generate the public key check value/thumb print are supported. This property is null if not supported.
keyManagement/rsaKeyCheckMode/sha1 sha1 is supported as defined in [Ref. keymanagement-2]. default: false
keyManagement/rsaKeyCheckMode/sha256 sha256 is supported as defined in ISO/IEC 10118-3:2004 [Ref. keymanagement-7] and FIPS 180-2 [Ref. keymanagement-8]. default: false
keyManagement/signatureScheme Specifies which capabilities are supported by the Signature scheme. This property is null if not supported.
keyManagement/signatureScheme/randomNumber Specifies if the service returns a random number from the KeyManagement.StartKeyExchange command within the RSA Signature Scheme. default: false
keyManagement/signatureScheme/exportDeviceId Specifies if the service supports exporting the device Security Item within the RSA Signature Scheme. default: false
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}. default: false
keyManagement/emvImportSchemes Identifies the supported EMV Import Scheme(s). This property is null if not supported. default: null
keyManagement/emvImportSchemes/plainCA A plain text CA public key is imported with no verification. default: false

Properties
keyManagement/emvImportSchemes/chksumCA A plain text CA public key is imported using the EMV 2000 verification algorithm. See [Ref. keymanagement-3]. default: false
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]. default: false
keyManagement/emvImportSchemes/issuer An Issuer public key is imported as defined in EMV 2000 Book II, [Ref. keymanagement-3]. default: false
keyManagement/emvImportSchemes/icc An ICC public key is imported as defined in EMV 2000 Book II, [Ref. keymanagement-3]. default: false
keyManagement/emvImportSchemes/iccPin An ICC PIN public key is imported as defined in EMV 2000 Book II, [Ref. keymanagement-3]. default: false
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.. default: false
keyManagement/keyBlockImportFormats Supported key block formats. This property is null if not supported. default: null
keyManagement/keyBlockImportFormats/A Supports X9.143 key block version ID A. default: false
keyManagement/keyBlockImportFormats/B Supports X9.143 key block version ID B. default: false
keyManagement/keyBlockImportFormats/C Supports X9.143 key block version ID C. default: false
keyManagement/keyBlockImportFormats/D Supports X9.143 key block version ID D. default: false
keyManagement/keyImportThroughParts Specifies whether the device is capable of importing keys in multiple parts. default: false
keyManagement/desKeyLength Specifies which DES key lengths are supported. This property is null if not supported. default: null
keyManagement/desKeyLength/single 8 byte DES keys are supported. default: false

Properties
keyManagement/desKeyLength/double 16 byte DES keys are supported. default: false
keyManagement/desKeyLength/triple 24 byte DES keys are supported. default: false
keyManagement/certificateTypes Specifies supported certificate types. This property is null if not supported. default: null
keyManagement/certificateTypes/encKey Supports the device public encryption certificate. default: false
keyManagement/certificateTypes/verificationKey Supports the device public verification certificate. default: false
keyManagement/certificateTypes/hostKey Supports the Host public certificate. default: false
keyManagement/loadCertOptions Specifying the options supported by the KeyManagement.LoadCertificate command. This property is null if not supported. Property value constraints: `minProperties: 1` default: null
keyManagement/loadCertOptions/certHost (example name) Specifies a supported signer. The following signers are possible. • `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]. Property name constraints: `pattern: ^(certHost|sigHost|hl|certHostTr34|caTr34|hlTr34|hlTr34)$`
keyManagement/loadCertOptions/certHost/newHost Load a new Host certificate, where one has not already been loaded. default: false
keyManagement/loadCertOptions/certHost/replaceHost Replace (or rebind) the device to a new Host certificate, where the new Host certificate is signed by *signer*. default: false
keyManagement/crklLoadOptions Supported options to load the Key Transport Key using the Certificate Remote Key Loading protocol. This property is null if not supported. default: null

Properties
keyManagement/crklLoadOptions/noRandom Import a Key Transport Key without generating and using a random number. default: false
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. default: false
keyManagement/crklLoadOptions/random Import a Key Transport Key by generating and using a random number. default: false
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. default: false
keyManagement/symmetricKeyManagementMethods Specifies the Symmentric Key Management modes. This property is null if not supported. default: null
keyManagement/symmetricKeyManagementMethods/fixedKey This method of key management uses fixed keys for transaction processing. default: false
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. default: false
keyManagement/symmetricKeyManagementMethods/tdesDukpt This method uses TDES Derived Unique Key Per Transaction (see [Ref. keymanagement-10]). default: false
keyManagement/keyAttributes 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` - Base Derivation Key (BDK).
- `B1` - Initial DUKPT Key.
- `B2` - Base Key Variant Key (deprecated).
- `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` - MAC Key, ISO 16609 MAC algorithm 1 (using TDEA).
- `M1` - MAC Key, ISO 9797-1 MAC Algorithm 1.
- `M2` - MAC Key, ISO 9797-1 MAC Algorithm 2.
- `M3` - MAC Key, ISO 9797-1 MAC Algorithm 3.
- `M4` - MAC Key, ISO 9797-1 MAC Algorithm 4.
- `M5` - MAC Key, ISO 9797-1:2011 MAC Algorithm 5.
- `M6` - MAC Key, ISO 9797-1:2011 MAC Algorithm 5 / CMAC.
- `M7` - HMAC Key.
- `M8` - MAC Key, 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, non-ANSI X9.24 Key.
- `V0` - PIN Verification Key, PVK, other algorithm.
- `V1` - PIN Verification Key, IBM 3624.
- `V2` - PIN Verification Key, VISA PVV.
- `V3` - PIN Verification Key, ANSI X9-132 algorithm 1.
- `V4` - PIN Verification Key, ANSI 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]$
```

Property value constraints:

```
minProperties: 1
```

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 (Note that this is included for backwards compatibility).
• H - HMAC (specify the underlying hash algorithm in optional field).
• R - RSA.
• T - Triple DEA (TDEA).
• 0 – 9 - These numeric values are reserved for proprietary use.
Property name constraints:
`pattern: ^[0-9ADHRT]$`
Property value constraints:
`minProperties: 1`

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/Wrap and Decrypt/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]$` default: null

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 null 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]$
```

default: null

Properties
keyManagement/decryptAttributes Attributes supported by the KeyManagement.ImportKey command for the key used to decrypt or unwrap the key being imported. default: null
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 or more of the following properties must 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. keymanagement-11]). • `xts` - The XTS method defined in NIST SP800-38E (See [Ref. keymanagement-12]). If the algorithm is 'R' then one or more of the following properties must be true. • `rsaesPkcs1V15` - Use the RSAES_PKCS1-v1.5 algorithm. • `rsaesOaep` - Use the RSAES OAEP algorithm. Property value constraints: `minProperties: 1`
keyManagement/decryptAttributes/A/decryptMethod/ecb The ECB encryption method is supported. default: false
keyManagement/decryptAttributes/A/decryptMethod/cbc The CBC encryption method is supported. default: false
keyManagement/decryptAttributes/A/decryptMethod/cfb The CFB encryption method is supported. default: false
keyManagement/decryptAttributes/A/decryptMethod/ofb The OFB encryption method is supported. default: false
keyManagement/decryptAttributes/A/decryptMethod/ctr The CTR method is supported and defined in NIST SP800-38A (See [Ref. 11]). default: false
keyManagement/decryptAttributes/A/decryptMethod/xts The XTS method is supported and defined in NIST SP800-38E (See [Ref. keymanagement-12]). default: false

Properties
keyManagement/decryptAttributes/A/decryptMethod/rsaesPkcs1V15 The RSAES-PKCS1-v1.5 algorithm is supported. default: false
keyManagement/decryptAttributes/A/decryptMethod/rsaesOaep The RSAES-OAEP algorithm is supported. default: false
keyManagement/verifyAttributes Attributes supported by the KeyManagement.ImportKey for the key used for verification before importing the key. default: null

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]$`
Property value constraints:
`minProperties: 1`

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]$`
Property value constraints:
`minProperties: 1`

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]$`

Properties
keyManagement/verifyAttributes/M0/T/V/cryptoMethod This parameter specifies the cryptographic method that will be used with encryption algorithm. If the algorithm is 'A', 'D', or 'T' and the key usage is a MAC usage (i.e. 'M1'), then all properties are false. If the algorithm is 'A', 'D', or 'T' and the key usage is '00', then one of properties must be set true. • `kcvNone` - There is no key check value (KCV) verification required. • `kcvSelf` - The KCV is created by an encryption of the key with itself. • `kcvZero` - The 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 There is no key check value (KCV) verification required. default: false
keyManagement/verifyAttributes/M0/T/V/cryptoMethod/kcvSelf The key check value (KCV) is created by an encryption of the key with itself. default: false
keyManagement/verifyAttributes/M0/T/V/cryptoMethod/kcvZero The key check value (KCV) is created by encrypting a zero value with the key. default: false
keyManagement/verifyAttributes/M0/T/V/cryptoMethod/sigNone The No signature algorithm specified. No signature verification will take place. default: false
keyManagement/verifyAttributes/M0/T/V/cryptoMethod/rsassaPkcs1V15 The RSASSA-PKCS1-v1.5 algorithm. default: false
keyManagement/verifyAttributes/M0/T/V/cryptoMethod/rsassaPss The RSASSA-PSS algorithm. default: false
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. default: false
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]. default: false
keyboard Capability information for XFS4IoT services implementing the Keyboard interface. This will be null if the Keyboard interface is not supported. default: null

Properties
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. This property is null if not supported. default: null
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. default: false
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. default: false
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. default: false
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. default: false
keyboard/etsCaps Specifies the capabilities of the Encrypting Touch Screen device. This property is null if not supported. default: null
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` default: 0
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` default: 0
keyboard/etsCaps/xSize Specifies the width of the Encrypting Touch Screen in virtual screen coordinates. Property value constraints: `minimum: 0` default: 0
keyboard/etsCaps/ySize Specifies the height of the Encrypting Touch Screen in virtual screen coordinates. Property value constraints: `minimum: 0` default: 0

Properties
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` default: 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` default: 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. This property is null if not supported. default: null
keyboard/etsCaps/float/x Specifies that the device will randomly shift the layout in a horizontal direction. default: false
keyboard/etsCaps/float/y Specifies that the device will randomly shift the layout in a vertical direction. default: false
textTerminal Capability information for XFS4IoT services implementing the TextTerminal interface. This will be null if the TextTerminal interface is not supported. default: null
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. Property value constraints: `minItems: 1`
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`
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.

Properties
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 null if the Printer interface is not supported. default: null
printer/type Specifies the type(s) of the physical device driven by the logical service.
printer/type/receipt The device is a receipt printer. default: false
printer/type/passbook The device is a passbook printer. default: false
printer/type/journal The device is a journal printer. default: false
printer/type/document The device is a document printer. default: false
printer/type/scanner The device is a scanner that may have printing capabilities. default: false
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. default: false
printer/resolution/medium The device can print medium resolution. default: false
printer/resolution/high The device can print high resolution. default: false
printer/resolution/veryHigh The device can print very high resolution. default: false
printer/readForm Specifies whether the device can read data from media. This property is null if the device can not read data. default: null
printer/readForm/ocr Device has OCR capability. default: false

Properties
printer/readForm/micr Device has MICR capability. default: false
printer/readForm/msf Device has MSF capability. default: false
printer/readForm/barcode Device has Barcode capability. default: false
printer/readForm/pageMark Device has Page Mark capability. default: false
printer/readForm/readImage Device has imaging capability. default: false
printer/readForm/readEmptyLine Device has capability to detect empty print lines for passbook printing. default: false
printer/writeForm Specifies whether the device can write data to the media.
printer/writeForm/text Device has Text capability. default: false
printer/writeForm/graphics Device has Graphics capability. default: false
printer/writeForm/stamp Device has stamping capability. default: false
printer/extents Specifies whether the device is able to measure the inserted media. This property is null if the device is unable to measure inserted media.
printer/extents/horizontal Device has horizontal size detection capability. default: false
printer/extents/vertical Device has vertical size detection capability. default: false
printer/control Specifies the manner in which media can be controlled. Property value constraints: `minProperties: 1`
printer/control/eject Device can eject media. default: false

Properties
printer/control/perforate Device can perforate media. default: false
printer/control/cut Device can cut media. default: false
printer/control/skip Device can skip to mark. default: false
printer/control/flush Device can be sent data that is buffered internally, and flushed to the printer on request. default: false
printer/control/retract Device can retract media under application control. default: false
printer/control/stack Device can stack media items before ejecting as a bundle. default: false
printer/control/partialCut Device can partially cut the media. default: false
printer/control/alarm Device can ring a bell, beep or otherwise sound an audible alarm. default: false
printer/control/pageForward Capability to turn one page forward. default: false
printer/control/pageBackward Capability to turn one page backward. default: false
printer/control/turnMedia Device can turn inserted media. default: false
printer/control/stamp Device can stamp on media. default: false
printer/control/park Device can park a document into the parking station. default: false
printer/control/expel Device can expel media out of the exit slot. default: false
printer/control/ejectToTransport Device can move media to a position on the transport just behind the exit slot. default: false

Properties
printer/control/rotate180 Device can rotate media 180 degrees in the printing plane. default: false
printer/control/clearBuffer The Service can clear buffered data. default: false
printer/maxMediaOnStacker Specifies the maximum number of media items that the stacker can hold. Property value constraints: `minimum: 0` default: 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. default: false
printer/multiPage Specifies whether the device is able to support multiple page print jobs. default: false
printer/paperSources Specifies the paper sources available for this printer.
printer/paperSources/upper The upper paper source. default: false
printer/paperSources/lower The lower paper source. default: false
printer/paperSources/external The external paper source. default: false
printer/paperSources/aux The auxiliary paper source. default: false
printer/paperSources/aux2 The second auxiliary paper source. default: false
printer/paperSources/park The parking station. default: false
printer/paperSources/exampleProperty1 (example name) The vendor specific paper source. Property name constraints: `pattern: ^[a-zA-Z]([a-zA-Z0-9]*)$` default: false
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. default: false

Properties
printer/retractBins Specifies the number of retract bins. Property value constraints: `minimum: 0` default: 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 null if there are no retract bins. Property value constraints: `minimum: 0` default: null
printer/imageType Specifies the image format supported by this device. This will be null if the device is unable to scan images. default: null
printer/imageType/tif The device can return scanned images in TIFF 6.0 format. default: false
printer/imageType/wmf The device can return scanned images in WMF (Windows Metafile) format. default: false
printer/imageType/bmp The device can return scanned images in Windows BMP format. default: false
printer/imageType/jpg The device can return scanned images in JPG format. default: false
printer/frontImageColorFormat Specifies the front image color formats supported by this device. This will be null if the device is unable to scan front images. default: null
printer/frontImageColorFormat/binary The device can return scanned images in binary (image contains two colors, usually the colors black and white). default: false
printer/frontImageColorFormat/grayscale The device can return scanned images in gray scale (image contains multiple gray colors). default: false
printer/frontImageColorFormat/full The device can return scanned images in full color (image contains colors like red, green, blue etc.). default: false
printer/backImageColorFormat Specifies the back image color formats supported by this device. This will be null if the device is unable to scan back images. default: null
printer/imageSource Specifies the source for the read image command supported by this device. This will be null if the device does not support reading images. default: null

Properties
printer/imageSource/imageFront The device can scan the front image of the document. default: false
printer/imageSource/imageBack The device can scan the back image of the document. default: false
printer/dispensePaper Specifies whether the device is able to dispense paper. default: false
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 will be null if the Service does not support the *Printer.PrintNative* command. default: null
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. default: false
printer/autoRetractPeriod Specifies the number of seconds before the device will automatically retract the presented media. If the command that generated the media is still active when the media is automatically retracted, the command will complete with an error. If the device does not retract media automatically this value is 0. Property value constraints: `minimum: 0` default: 0
printer/retractToTransport Specifies whether the device is able to retract the previously ejected media to the transport. default: false
printer/coercivityType Specifies the form write modes supported by this device. This will be null if the device is unable to write magnetic stripes. default: null
printer/coercivityType/low This device can write the magnetic stripe by low coercivity mode. default: false
printer/coercivityType/high This device can write the magnetic stripe by high coercivity mode. default: false
printer/coercivityType/auto The Service or the device is capable of automatically determining whether low or high coercivity magnetic stripe should be written. default: false
printer/controlPassbook Specifies how the passbook can be controlled with the Printer.ControlPassbook command. This will be null if the command is not supported. default: null

Properties
printer/controlPassbook/turnForward The device can turn forward multiple pages of the passbook. default: false
printer/controlPassbook/turnBackward The device can turn backward multiple pages of the passbook. default: false
printer/controlPassbook/closeForward The device can close the passbook forward. default: false
printer/controlPassbook/closeBackward The device can close the passbook backward. default: false
printer/printSides Specifies on which sides of the media this device can print as one of the following values. This will be null if 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. default: null
barcodeReader Capability information for XFS4IoT services implementing the BarcodeReader interface. This will be null if the BarcodeReader interface is not supported. default: null
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 null if the supported barcode symbologies can not be determined. default: null
barcodeReader/symbologies/ean128 GS1-128 default: false
barcodeReader/symbologies/ean8 EAN-8 default: false
barcodeReader/symbologies/ean8_2 EAN-8 with 2 digit add-on default: false
barcodeReader/symbologies/ean8_5 EAN-8 with 5 digit add-on default: false
barcodeReader/symbologies/ean13 EAN-13 default: false
barcodeReader/symbologies/ean13_2 EAN-13 with 2 digit add-on default: false

Properties
barcodeReader/symbologies/ean13_5 EAN-13 with 5 digit add-on default: false
barcodeReader/symbologies/jan13 JAN-13 default: false
barcodeReader/symbologies/upcA UPC-A default: false
barcodeReader/symbologies/upcE0 UPC-E default: false
barcodeReader/symbologies/upcE0_2 UPC-E with 2 digit add-on default: false
barcodeReader/symbologies/upcE0_5 UPC-E with 5 digit add-on default: false
barcodeReader/symbologies/upcE1 UPC-E with leading 1 default: false
barcodeReader/symbologies/upcE1_2 UPC-E with leading 1and 2 digit add-on default: false
barcodeReader/symbologies/upcE1_5 UPC-E with leading 1and 5 digit add-on default: false
barcodeReader/symbologies/upcA_2 UPC-A with2 digit add-on default: false
barcodeReader/symbologies/upcA_5 UPC-A with 5 digit add-on default: false
barcodeReader/symbologies/codabar CODABAR (NW-7) default: false
barcodeReader/symbologies/itf Interleaved 2 of 5 (ITF) default: false
barcodeReader/symbologies/code11 CODE 11 (USD-8) default: false
barcodeReader/symbologies/code39 CODE 39 default: false

Properties
barcodeReader/symbologies/code49 CODE 49 default: false
barcodeReader/symbologies/code93 CODE 93 default: false
barcodeReader/symbologies/code128 CODE 128 default: false
barcodeReader/symbologies/msi MSI default: false
barcodeReader/symbologies/plessey PLESSEY default: false
barcodeReader/symbologies/std2Of5 STANDARD 2 of 5 (INDUSTRIAL 2 of 5 also) default: false
barcodeReader/symbologies/std2Of5Iata STANDARD 2 of 5 (IATA Version) default: false
barcodeReader/symbologies/pdf417 PDF-417 default: false
barcodeReader/symbologies/microPdf417 MICROPDF-417 default: false
barcodeReader/symbologies/dataMatrix GS1 DataMatrix default: false
barcodeReader/symbologies/maxiCode MAXICODE default: false
barcodeReader/symbologies/codeOne CODE ONE default: false
barcodeReader/symbologies/channelCode CHANNEL CODE default: false
barcodeReader/symbologies/telepenOriginal Original TELEPEN default: false
barcodeReader/symbologies/telepenAim AIM version of TELEPEN default: false

Properties
barcodeReader/symbologies/rss GS1 DataBar™ default: false
barcodeReader/symbologies/rssExpanded Expanded GS1 DataBar™ default: false
barcodeReader/symbologies/rssRestricted Restricted GS1 DataBar™ default: false
barcodeReader/symbologies/compositeCodeA Composite Code A Component default: false
barcodeReader/symbologies/compositeCodeB Composite Code B Component default: false
barcodeReader/symbologies/compositeCodeC Composite Code C Component default: false
barcodeReader/symbologies/posiCodeA Posicode Variation A default: false
barcodeReader/symbologies/posiCodeB Posicode Variation B default: false
barcodeReader/symbologies/triopticCode39 Trioptic Code 39 default: false
barcodeReader/symbologies/codablockF Codablock F default: false
barcodeReader/symbologies/code16K Code 16K default: false
barcodeReader/symbologies/qrCode QR Code default: false
barcodeReader/symbologies/aztec Aztec Codes default: false
barcodeReader/symbologies/ukPost UK Post default: false
barcodeReader/symbologies/planet US Postal Planet default: false

Properties
barcodeReader/symbologies/postnet US Postal Postnet default: false
barcodeReader/symbologies/canadianPost Canadian Post default: false
barcodeReader/symbologies/netherlandsPost Netherlands Post default: false
barcodeReader/symbologies/australianPost Australian Post default: false
barcodeReader/symbologies/japanesePost Japanese Post default: false
barcodeReader/symbologies/chinesePost Chinese Post default: false
barcodeReader/symbologies/koreanPost Korean Post default: false
biometric Capability information for XFS4IoT services implementing the Biometrics interface. This will be null if the Biometrics interface is not supported. default: null
biometric/type Specifies the type of biometric device.
biometric/type/facialFeatures The biometric device supports facial recognition scanning. default: false
biometric/type/voice The biometric device supports voice recognition. default: false
biometric/type/fingerprint The biometric device supports fingerprint scanning. default: false
biometric/type/fingerVein The biometric device supports finger vein scanning. default: false
biometric/type/iris The biometric device supports iris scanning. default: false
biometric/type/retina The biometric device supports retina scanning. default: false

Properties
biometric/type/handGeometry The biometric device supports hand geometry scanning. default: false
biometric/type/thermalFace The biometric device supports thermal face image scanning. default: false
biometric/type/thermalHand The biometric device supports thermal hand image scanning. default: false
biometric/type/palmVein The biometric device supports palm vein scanning. default: false
biometric/type/signature The biometric device supports signature scanning. default: false
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]. default: false
biometric/dataFormats/isoFmd ISO FMD template format [Ref. biometric-4]. default: false
biometric/dataFormats/ansiFid Raw ANSI FID format [Ref. biometric-1]. default: false
biometric/dataFormats/ansiFmd ANSI FMD template format [Ref. biometric-2]. default: false
biometric/dataFormats/qso Raw QSO image format. default: false
biometric/dataFormats/wso WSQ image format. default: false

Properties
biometric/dataFormats/reservedRaw1 Reserved for a vendor-defined Raw format. default: false
biometric/dataFormats/reservedTemplate1 Reserved for a vendor-defined Template format. default: false
biometric/encryptionAlgorithms Supported encryption algorithms. This property is null if no encryption algorithms are supported. default: null
biometric/encryptionAlgorithms/ecb Triple DES with Electronic Code Book. default: false
biometric/encryptionAlgorithms/cbc Triple DES with Cipher Block Chaining. default: false
biometric/encryptionAlgorithms/cfb Triple DES with Cipher Feed Back. default: false
biometric/encryptionAlgorithms/rsa RSA Encryption. default: false
biometric/storage Indicates whether or not biometric template data can be stored securely. This property is null if biometric template data is not stored in the device. default: null
biometric/storage/secure Biometric template data is securely stored as encrypted data. default: false
biometric/storage/clear Biometric template data is stored unencrypted in the device. default: false
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. This property is null if persistence is entirely under device control and cannot be set. default: null
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.Read is requested again. This captured biometric data can also be explicitly cleared using the Biometric.Clear or Biometric.Reset. default: false
biometric/persistenceModes/clear Captured biometric data will not persist. Once the data has been either returned in the Biometric.Read or used by the Biometric.Match, then the data is cleared from the device. default: false

Properties
biometric/matchSupported Specifies if matching is supported using the Biometric.Match and/or Biometric.SetMatch command. This property is null 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. default: null
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. default: false
biometric/scanModes/match The Biometric.Read can be used to scan data for a match operation using the Biometric.Match. default: false
biometric/compareModes Specifies the type of match operations. This property is null if the device does not support matching. default: null
biometric/compareModes/verify The biometric data can be compared as a one to one verification operation. default: false
biometric/compareModes/identity The biometric data can be compared as a one to many identification operation. default: false
biometric/clearData Specifies the type of data that can be cleared from storage using the Biometric.Clear or Biometric.Reset command. This property is null if the device does not support clearing data from storage using commands. default: null
biometric/clearData/scannedData Raw image data that has been scanned using the Biometric.Read can be cleared. default: false
biometric/clearData/importedData Template data that was imported using the Biometric.Import can be cleared. default: false
biometric/clearData/setMatchedData Match criteria data that was set using the Biometric.Match can be cleared. default: false
camera Capability information for XFS4IoT services implementing the Camera interface. This will be null if the Camera interface is not supported. default: null
camera/cameras Specifies whether cameras are available.

Properties
camera/cameras/room Specifies whether the camera that monitors the whole self-service area is available. default: false
camera/cameras/person Specifies whether the camera that monitors the person standing in front of the self-service is available. default: false
camera/cameras/exitSlot Specifies whether the camera that monitors the exit slot(s) of the self-service machine is available. default: false
camera/cameras/vendorSpecificCamera (example name) Allows vendor specific cameras to be reported. default: false
camera/maxPictures Specifies the maximum number of pictures that can be stored on the recording media. This property is null if not applicable. Property value constraints: `minimum: 0` default: null
camera/camData Specifies whether the methods are supported for adding data to the picture. If null, no data can be added to the picture. default: null
camera/camData/autoAdd Specifies whether data can be added automatically to the picture. default: false
camera/camData/manAdd Specifies whether data can be added manually to the picture using Camera.TakePicture.camData. default: false
camera/maxDataLength Specifies the maximum length of the data that is displayed on the photo. This property is null if not applicable. Property value constraints: `minimum: 0` default: null
lights Capability information for XFS4IoT services implementing the Lights interface. This will be null if the Lights interface is not supported. default: null
lights/cardReader Card Unit Light. default: null
lights/cardReader/flashRate Indicates the light flash rate.
lights/cardReader/flashRate/off The light can be turned off. default: false

Properties
lights/cardReader/flashRate/slow The light can flash slowly. default: false
lights/cardReader/flashRate/medium The light can flash medium frequency. default: false
lights/cardReader/flashRate/quick The light can flash quickly. default: false
lights/cardReader/flashRate/continuous The light can flash continuous (steady). default: true
lights/cardReader/color Indicates the light color.
lights/cardReader/color/red The light can be red. default: false
lights/cardReader/color/green The light can be green. default: false
lights/cardReader/color/yellow The light can be yellow. default: false
lights/cardReader/color/blue The light can be blue. default: false
lights/cardReader/color/cyan The light can be cyan. default: false
lights/cardReader/color/magenta The light can be magenta. default: false
lights/cardReader/color/white The light can be white. default: false
lights/cardReader/direction Indicates the light direction. This property is null if not applicable. default: null
lights/cardReader/direction/entry The light can indicate entry. default: false
lights/cardReader/direction/exit The light can indicate exit. default: false

Properties
lights/cardReader/position Indicates the light position. This property is null if not applicable. default: null
lights/cardReader/position/left The left position. default: false
lights/cardReader/position/right The right position. default: false
lights/cardReader/position/center The center position. default: false
lights/cardReader/position/top The top position. default: false
lights/cardReader/position/bottom The bottom position. default: false
lights/cardReader/position/front The front position. default: false
lights/cardReader/position/rear The rear position. default: false
lights/pinPad Pin Pad Light. default: null
lights/notesDispenser Notes Dispenser Light. default: null
lights/coinDispenser Coin Dispenser Light. default: null
lights/receiptPrinter Receipt Printer Light. default: null
lights/passbookPrinter Passbook Printer Light. default: null
lights/envelopeDepository Envelope Depository Light. default: null
lights/checkUnit Check Unit Light. default: null

Properties
lights/billAcceptor Bill Acceptor Light. default: null
lights/envelopeDispenser Envelope Dispenser Light. default: null
lights/documentPrinter Document Printer Light. default: null
lights/coinAcceptor Coin Acceptor Light. default: null
lights/scanner Scanner Light. default: null
lights/contactless Contactless Reader Light. default: null
lights/cardReader2 Card Reader 2 Light. default: null
lights/notesDispenser2 Notes Dispenser 2 Light. default: null
lights/billAcceptor2 Bill Acceptor 2 Light. default: null
lights/statusGood Status indicator light - Good. default: null
lights/statusWarning Status indicator light - Warning. default: null
lights/statusBad Status indicator light - Bad. default: null
lights/statusSupervisor Status indicator light - Supervisor. default: null
lights/statusInService Status indicator light - In Service. default: null
lights/fasciaLight Fascia light. default: null

Properties
lights/vendorSpecificLight (example name) Additional vendor specific lights. default: null
auxiliaries Capability information for XFS4IoT services implementing the Auxiliaries interface. This will be null if the Auxiliaries interface is not supported. default: null
auxiliaries/operatorSwitch Specifies which states the Operator Switch can be set to. If not available, this property is null. default: null
auxiliaries/operatorSwitch/run The switch can be set in Run mode. default: false
auxiliaries/operatorSwitch/maintenance The switch can be set in Maintenance mode. default: false
auxiliaries/operatorSwitch/supervisor The switch can be set in Supervisor mode. default: false
auxiliaries/tamperSensor Specifies whether the Tamper Sensor for the terminal is available. default: false
auxiliaries/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. Null if not applicable. default: null
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

Properties
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/consumerDisplaySensor 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. Null if not applicable. default: null
auxiliaries/handsetSensor/manual The Handset is available and it supports manual mode. default: false
auxiliaries/handsetSensor/auto The Handset is available and it supports auto mode. default: false
auxiliaries/handsetSensor/semiAuto The Handset is available and it supports semi-auto mode. default: false
auxiliaries/handsetSensor/microphone The Handset is available and contains an embedded microphone for audio input. default: false
auxiliaries/headsetMicrophoneSensor Specifies whether the Microphone Jack is present, and if so, which modes it supports. If the *enhancedAudio* capability indicates the presence of a bi-directional Audio Jack then both sensors reference the same physical jack. Null if not applicable. default: null
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

Properties
auxiliaries/fasciaMicrophoneSensor Specifies whether a Fascia Microphone (for public audio input) is present. default: false
auxiliaries/cabinetDoor Specifies whether the Cabinet Door is available, and if so, which states it supports. If there are multiple Cabinet Doors available, use appropriate position of Cabinet Door. *frontCabinet*, *rearCabinet*, *leftCabinet* or *rightCabinet* properties. Null if not applicable. default: null
auxiliaries/cabinetDoor/closed Specifies that the door can report the closed state. default: false
auxiliaries/cabinetDoor/open Specifies that the door can report the open state. default: false
auxiliaries/cabinetDoor/locked Specifies that the door can report the locked state. default: false
auxiliaries/cabinetDoor/bolted Specifies that the door can report the bolted state. default: false
auxiliaries/cabinetDoor/tampered Specifies that the door can report the tampered state. default: false
auxiliaries/safeDoor Specifies whether the Safe Door is available, and if so, which states it supports. Null if not applicable. default: null
auxiliaries/vandalShield Specifies the states the Vandal Shield can support, if available. Null if not applicable. default: null
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

Properties
auxiliaries/frontCabinet Specifies whether at least one Front Cabinet Door is available, and if so, which states they support. Null if not applicable. default: null
auxiliaries/rearCabinet Specifies whether at least one Rear Cabinet Door is available, and if so, which states they support. Null if not applicable. default: null
auxiliaries/leftCabinet Specifies whether at least one left Cabinet Door is available, and if so, which states they support. Null if not applicable. default: null
auxiliaries/rightCabinet Specifies whether at least one right Cabinet Door is available, and if so, which states they support. Null if not applicable. default: null
auxiliaries/openCloseIndicator Specifies whether the Open/Closed Indicator is available. default: false
auxiliaries/audio Specifies whether the Audio Indicator device is available. default: false
auxiliaries/heating Specifies whether the Internal Heating device is available. default: false
auxiliaries/consumerDisplayBacklight Specifies whether the Consumer Display Backlight is available. default: false
auxiliaries/signageDisplay Specifies whether the Signage Display is available. default: false

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

auxiliaries/ups Specifies what states the UPS device supports. Null if not applicable. default: null
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

Properties
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. Null if not applicable. default: null
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/enhancedMicrophoneControl 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. Null if not applicable. default: null
auxiliaries/enhancedMicrophoneControl/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/enhancedMicrophoneControl/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. Null if not applicable.
Property value constraints:
`minimum: 1`
`maximum: 1000`
default: null

auxiliaries/autoStartupMode Specifies which modes of the auto start-up control are supported. Null if not applicable. default: null
auxiliaries/autoStartupMode/specific The device supports one-time auto start-up on a specific date at a specific time. default: false

Properties
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 null if the Vendor Application interface is not supported. default: null
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 functionality definition is left to the vendor. If no access levels are supported this property will be null. default: null
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. default: false
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. default: false
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. default: false

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 2.0)	Type	Required
{		
"commands": {	object, null	
"CardReader.ReadRawData": 1,	integer	
"CardReader.Move": See commands/CardReader.ReadRawData	integer	
},		
"events": {	object, null	
"CardReader.MediaInsertedEvent": 1,	integer	
"CardReader.MediaRemovedEvent": See events/CardReader.MediaInsertedEvent	integer	
}		
}		

Properties

commands

The commands for which a version is being set.

Property value constraints:

```
minProperties: 1
```

default: null

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
```

default: null

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 2.0)
This message does not define any properties.

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 2.0)	Type	Required
{		
"requestIds": [1, 2]	array (integer), null	
}		

Properties

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 null, 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:

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

default: null

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "noMatchingRequestIDs"	string, null	
}		

Properties

errorCode

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

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

default: null

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 2.0)	Type	Required
{		
"maxPowerSaveRecoveryTime": 5	integer	✓
}		

Properties

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 2.0)
This message does not define any properties.

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 2.0)	Type	Required
`{`		
`"state": "active",`	string	✓
`"transactionID": "Example transaction ID"`	string, null	
`}`		

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.

if *state* is *inactive*, this property:

- Is ignored in Common.SetTransactionState
- Is null in Common.GetTransactionState.

default: null

Completion Message

Payload (version 2.0)
`This message does not define any properties.`

Event Messages

None

4.1.7 Common.GetTransactionState

This command can be used to get the transaction state.

Command Message

Payload (version 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"state": "active",`	string	✓
`"transactionID": "Example transaction ID"`	string, null	
`}`		

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.

if *state* is *inactive*, this property:

- Is ignored in Common.SetTransactionState
- Is null in Common.GetTransactionState.

default: null

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 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"commandNonce": "646169ECDD0E440C2CECC8DDD7C27C22"`	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}$|^[0-9]*$
```

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 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)
`This message does not define any properties.`

Event Messages

None

4.2 Unsolicited Messages

4.2.1 Common.StatusChangedEvent

This event reports that a change of 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 null.

Unsolicited Message

Payload (version 2.0)	Type	Required
{		
"common": {	object, null	
"device": "online",	string, null	
"devicePosition": "notInPosition",	string, null	
"powerSaveRecoveryTime": 10,	integer, null	
"antiFraudModule": "ok",	string, null	
"exchange": "active",	string, null	
"endToEndSecurity": "enforced"	string, null	
},		
"cardReader": {	object, null	
"media": "unknown",	string, null	
"security": "notReady",	string, null	
"chipPower": "unknown",	string, null	
"chipModule": "ok",	string, null	
"magWriteModule": "ok",	string, null	
"frontImageModule": "ok",	string, null	
"backImageModule": "ok"	string, null	
},		
"cashAcceptor": {	object, null	
"intermediateStacker": "empty",	string, null	
"stackerItems": "customerAccess",	string, null	
"banknoteReader": "ok",	string, null	
"dropBox": true,	boolean, null	
"positions": [{	array (object), null	
"position": "inLeft",	string	✓
"shutter": "closed",	string, null	
"positionStatus": "empty",	string, null	
"transport": "ok",	string, null	
"transportStatus": "empty"	string, null	
}]		
},		
"cashDispenser": {	object, null	

Payload (version 2.0)	Type	Required
"intermediateStacker": "empty",	string, null	
"positions": [{	array (object), null	
"position": "outDefault",	string	
"shutter": "closed",	string, null	
"positionStatus": "empty",	string, null	
"transport": "ok",	string, null	
"transportStatus": "empty"	string, null	
}]		
},		
"cashManagement": {	object, null	
"dispenser": "ok",	string, null	
"acceptor": "ok"	string, null	
},		
"check": {	object, null	
"acceptor": "ok",	string, null	
"media": "present",	string, null	
"toner": "full",	string, null	
"ink": "full",	string, null	
"frontImageScanner": "ok",	string, null	
"backImageScanner": "ok",	string, null	
"mICRReader": "ok",	string, null	
"stacker": "empty",	string, null	
"rebuncher": "empty",	string, null	
"mediaFeeder": "notEmpty",	string, null	
"positions": {	object, null	
"input": {	object, null	
"shutter": "closed",	string, null	✓
"positionStatus": "empty",	string, null	
"transport": "ok",	string, null	
"transportMediaStatus": "empty",	string, null	
"jammedShutterPosition": "notJammed"	string, null	
},		
"output": See check/positions/input properties	object, null	
"refused": See check/positions/input properties	object, null	
}		
},		
"mixedMedia": {	object, null	
"modes": {	object	✓
"cashAccept": true,	boolean, null	

Payload (version 2.0)	Type	Required
"checkAccept": true	boolean, null	
}		
},		
"keyManagement": {	object, null	
"encryptionState": "ready",	string, null	
"certificateState": "unknown"	string, null	
},		
"keyboard": {	object, null	
"autoBeepMode": {	object	✓
"activeAvailable": false,	boolean, null	
"inactiveAvailable": false	boolean, null	
}		
},		
"textTerminal": {	object, null	
"keyboard": "on",	string, null	
"keyLock": "on",	string, null	
"displaySizeX": 0,	integer, null	
"displaySizeY": 0	integer, null	
},		
"printer": {	object, null	
"media": "unknown",	string, null	
"paper": {	object, null	
"upper": "unknown",	string, null	
"lower": "unknown",	string, null	
"external": "unknown",	string, null	
"aux": "unknown",	string, null	
"aux2": "unknown",	string, null	
"park": "unknown",	string, null	
"*vendorSpecificPaperSupply*": "unknown"	string, null	
},		
"toner": "unknown",	string, null	
"ink": "unknown",	string, null	
"lamp": "unknown",	string, null	
"retractBins": [{	array (object), null	
"state": "unknown",	string, null	
"count": 0	integer, null	
}],		
"mediaOnStacker": 7,	integer, null	
"paperType": {	object, null	

Payload (version 2.0)	Type	Required
"upper": "unknown",	string, null	
"lower": "unknown",	string, null	
"external": "unknown",	string, null	
"aux": "unknown",	string, null	
"aux2": "unknown",	string, null	
"park": "unknown",	string, null	
"*exampleProperty1*": "unknown",	string, null	
"*exampleProperty2*": See *printer/paperType/exampleProperty1*	string, null	
},		
"blackMarkMode": "unknown"	string, null	
},		
"barcodeReader": {	object, null	
"scanner": "on"	string	✓
},		
"biometric": {	object, null	
"subject": "present",	string, null	
"capture": false,	boolean, null	
"dataPersistence": "persist",	string, null	
"remainingStorage": 0	integer, null	
},		
"camera": {	object, null	
"media": {	object, null	
"room": "ok",	string, null	
"person": "ok",	string, null	
"exitSlot": "ok",	string, null	
"*vendorSpecificCameraMedia*": "ok"	string, null	
},		
"cameras": {	object, null	
"room": "ok",	string, null	
"person": "ok",	string, null	
"exitSlot": "ok",	string, null	
"*vendorSpecificCameraState*": See *camera/media/vendorSpecificCameraMedia*	string, null	
},		
"pictures": {	object, null	
"room": 0,	integer, null	
"person": 0,	integer, null	
"exitSlot": 0,	integer, null	
"*vendorSpecificCameraPictures*": 0	integer, null	
}		

Payload (version 2.0)	Type	Required
},		
"lights": {	object, null	
"cardReader": {	object, null	
"position": "left",	string	✓
"flashRate": "off",	string, null	
"color": "red",	string, null	
"direction": "entry"	string, null	
},		
"pinPad": See lights/cardReader properties	object, null	
"notesDispenser": See lights/cardReader properties	object, null	
"coinDispenser": See lights/cardReader properties	object, null	
"receiptPrinter": See lights/cardReader properties	object, null	
"passbookPrinter": See lights/cardReader properties	object, null	
"envelopeDepository": See lights/cardReader properties	object, null	
"checkUnit": See lights/cardReader properties	object, null	
"billAcceptor": See lights/cardReader properties	object, null	
"envelopeDispenser": See lights/cardReader properties	object, null	
"documentPrinter": See lights/cardReader properties	object, null	
"coinAcceptor": See lights/cardReader properties	object, null	
"scanner": See lights/cardReader properties	object, null	
"contactless": See lights/cardReader properties	object, null	
"cardReader2": See lights/cardReader properties	object, null	
"notesDispenser2": See lights/cardReader properties	object, null	
"billAcceptor2": See lights/cardReader properties	object, null	
"statusGood": See lights/cardReader properties	object, null	
"statusWarning": See lights/cardReader properties	object, null	
"statusBad": See lights/cardReader properties	object, null	
"statusSupervisor": See lights/cardReader properties	object, null	
"statusInService": See lights/cardReader properties	object, null	
"fasciaLight": See lights/cardReader properties	object, null	
"*vendorSpecificLight*": See *lights/cardReader* properties	object, null	
},		
"auxiliaries": {	object, null	
"operatorSwitch": "run",	string, null	
"tamperSensor": "on",	string, null	
"internalTamperSensor": "on",	string, null	
"seismicSensor": "on",	string, null	
"heatSensor": "on",	string, null	
"proximitySensor": "present",	string, null	
"ambientLightSensor": "veryDark",	string, null	

Payload (version 2.0)	Type	Required
"enhancedAudioSensor": "present",	string, null	
"bootSwitchSensor": "off",	string, null	
"consumerDisplaySensor": "off",	string, null	
"operatorCallButtonSensor": "off",	string, null	
"handsetSensor": "onTheHook",	string, null	
"headsetMicrophoneSensor": "present",	string, null	
"fasciaMicrophoneSensor": "off",	string, null	
"safeDoor": "closed",	string, null	
"vandalShield": "closed",	string, null	
"cabinetFrontDoor": "closed",	string, null	
"cabinetRearDoor": "closed",	string, null	
"cabinetLeftDoor": "closed",	string, null	
"cabinetRightDoor": "closed",	string, null	
"openClosedIndicator": "closed",	string, null	
"audio": {	object, null	
"rate": "on",	string, null	
"signal": "keypress"	string, null	
},		
"heating": "off",	string, null	
"consumerDisplayBacklight": "off",	string, null	
"signageDisplay": "off",	string, null	
"volume": 1,	integer, null	
"UPS": {	object, null	
"low": true,	boolean, null	
"engaged": false,	boolean, null	
"powering": false,	boolean, null	
"recovered": false	boolean, null	
},		
"audibleAlarm": "on",	string, null	
"enhancedAudioControl": "publicAudioManual",	string, null	
"enhancedMicrophoneControl": "publicAudioManual",	string, null	
"microphoneVolume": 1	integer, null	
},		
"vendorMode": {	object, null	
"device": "online",	string, null	
"service": "enterPending"	string, null	
},		
"vendorApplication": {	object, null	
"accessLevel": "notActive"	string	✓
}		

Payload (version 2.0)	Type	Required
}		

Properties

common

Status information common to all XFS4IoT services. May be null if none of the properties have changed.

default: null

common/device

Specifies the state of the device. This property is required in Common.Status, but may be null in Common.StatusChangedEvent if it has not changed. 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*.

default: null

common/devicePosition

Position of the device. This property is null in Common.Status if position status reporting is not supported, otherwise the 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.

default: null

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 the power saving mode has not been activated. This property is null in Common.Status if power save control is not supported.

Property value constraints:

```
minimum: 0
```

default: null

common/antiFraudModule

Specifies the state of the anti-fraud module if available. This property is null in Common.Status if there is no anti-fraud module, otherwise the 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.

default: null

Properties
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. This property is null in Common.Status if not supported, otherwise the following values are possible: • `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: null
common/endToEndSecurity Specifies the status of end to end security support on this device. This property is null in Common.Status if E2E security is not supported by this hardware and any command can be called without a token, otherwise the following values are possible. Also see Common.CapabilityProperties.endToEndSecurity. • `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: null
cardReader Status information for XFS4IoT services implementing the CardReader interface. This will be null if the CardReader interface is not supported. default: null
cardReader/media Specifies the transport/exit position media state. This property will be null if the capability to report media position is not supported by the device (e.g., a typical swipe reader or contactless chip card reader), otherwise one of the following values: • `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. default: null
cardReader/security Specifies the state of the security module. This property will be null if no security module is available, otherwise one of the following values: • `notReady` - The security module is not ready to process cards or is inoperable. • `open` - The security module is open and ready to process cards. default: null

Properties
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. This property will be null if the capability to report the state of the chip is not supported by the ID card unit device and will apply to contactless chip card readers, otherwise one of the following values: • `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. default: null
cardReader/chipModule Specifies the state of the chip card module reader. This property will be null if reporting the chip card module status is not supported, otherwise one of the following values: • `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. default: null
cardReader/magWriteModule Specifies the state of the magnetic card writer. This property will be null if reporting the magnetic card writing module status is not supported, otherwise one of the following values: • `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. default: null
cardReader/frontImageModule Specifies the state of the front image reader. This property will be null if reporting the front image reading module status is not supported, otherwise one of the following values: • `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. default: null
cardReader/backImageModule Specifies the state of the back image reader. This property will be null if reporting the back image reading module status is not supported, otherwise one of the following values: • `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. default: null
cashAcceptor Status information for XFS4IoT services implementing the CashAcceptor interface. This will be null if the CashAcceptor interface is not supported. default: null

Properties
cashAcceptor/intermediateStacker Supplies the state of the intermediate stacker. This property is null in Common.Status if the physical device has no intermediate stacker, otherwise 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. default: null
cashAcceptor/stackerItems This property informs the application whether items on the intermediate stacker have been in customer access. This property is null in Common.Status if the physical device has no intermediate stacker, otherwise 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. default: null
cashAcceptor/banknoteReader Supplies the state of the banknote reader. This property is null in Common.Status if the physical device has no banknote reader, otherwise 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. default: null
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 property 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. This property may be null if there is no drop box or its state has not changed in Common.StatusChangedEvent. default: null
cashAcceptor/positions Array of structures reporting status for each position from which items can be accepted. This may be null in Common.StatusChangedEvent if no position states have changed. default: null

Properties
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. This property is null in Common.Status if the physical position has no shutter, otherwise 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. default: null
cashAcceptor/positions/positionStatus The status of the input or output position. This property is null in Common.Status if the device is not capable of reporting whether items are at the position, otherwise 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. • `foreignItems` - Foreign items have been detected in the position. default: null
cashAcceptor/positions/transport Supplies the state of the transport mechanism. The transport is defined as any area leading to or from the position. This property is null in Common.Status if the device has no transport or transport state reporting is not supported, otherwise 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. default: null

Properties
cashAcceptor/positions/transportStatus Returns information regarding items which may be on the transport. If the device is a recycler device it is possible that the transport will not be empty due to a previous dispense operation. This property is null in Common.Status if the device has no transport or is not capable of reporting whether items are on the transport, otherwise 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. default: null
cashDispenser Status information for XFS4IoT services implementing the CashDispenser interface. This will be null if the CashDispenser interface is not supported. default: null
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. This property is null in Common.Status if the physical device has no intermediate stacker, otherwise the 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. default: null
cashDispenser/positions Array of structures for each position to which items can be dispensed or presented. This may be null in Common.StatusChangedEvent if no position states have changed. default: null
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. This property is null in Common.Status if the physical position has no shutter, otherwise the 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. default: null
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. This property is null in Common.Status if the device is not capable of reporting whether items are at the position, otherwise 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. default: null
cashDispenser/positions/transport Supplies the state of the transport mechanism. The transport is defined as any area leading to or from the position. This property is null in Common.Status if the device has no transport or transport state reporting is not supported, otherwise 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. default: null
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. This property is null in Common.Status if the device has no transport or is not capable of reporting whether items are on the transport, otherwise 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. default: null
cashManagement Status information for XFS4IoT services implementing the CashManagement interface. This will be null if the CashManagement interface is not supported. default: null

Properties
cashManagement/dispenser Supplies the state of the storage units for dispensing cash. This may be null in Common.Status if the device is not capable of dispensing cash, otherwise the 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. default: null
cashManagement/acceptor Supplies the state of the storage units for accepting cash. This may be null in Common.Status if the device is not capable of accepting cash, otherwise the 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 items are to be automatically retained within storage units (see retainAction), 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. default: null
check Status information for XFS4IoT services implementing the Check interface. This will be null if the Check interface is not supported. default: null
check/acceptor Supplies the state of the overall acceptor storage units. This may be null in Common.StatusChangedEvent if the state has not changed. The following values are possible: • `ok` - All storage units present are in a good state. • `state` - One or more of the storage units is in a high, full or inoperative condition. Items can still be accepted into at least one of the storage units. The status of the storage units can be obtained through the Storage.GetStorage command. • `stop` - Due to a storage unit problem accepting is impossible. No items can be accepted because all of the storage units are in a full or in an inoperative condition. • `unknown` - Due to a hardware error or other condition, the state of the storage units cannot be determined. default: null

Properties
check/media Specifies the state of the media. This may be null in Common.Status if the capability to report the state of the media is not supported by the device, otherwise the following values are possible: • `present` - Media is present in the device. • `notPresent` - Media is not present in the device. • `jammed` - Media is jammed in the device. • `unknown` - The state of the media cannot be determined with the device in its current state. • `position` - Media is at one or more of the input, output and refused positions. default: null
check/toner Specifies the state of the toner or ink supply or the state of the ribbon of the endorser. This may be null in Common.Status if the physical device does not support endorsing or the capability to report the status of the toner/ink is not supported by the device, otherwise the following values are possible: • `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. • `unknown` - Status of toner or ink supply or the ribbon cannot be determined with the device in its current state. default: null
check/ink Specifies the status of the stamping ink in the device. This may be null in Common.Status if the physical device does not support stamping or the capability to report the status of the stamp ink supply is not supported by the device, otherwise the following values are possible: • `full` - Ink supply in the device is full. • `low` - Ink supply in the device is low. • `out` - Ink supply in the device is empty. • `unknown` - Status of the stamping ink supply cannot be determined with the device in its current state. default: null
check/frontImageScanner Specifies the status of the image scanner that captures images of the front of the media items. This may be null in Common.Status if the physical device has no front scanner or the capability to report the status of the front scanner is not supported by the device, otherwise the following values are possible: • `ok` - The front scanner is OK. • `fading` - The front scanner performance is degraded. • `inoperative` - The front scanner is inoperative. • `unknown` - Status of the front scanner cannot be determined with the device in its current state. default: null
check/backImageScanner Specifies the status of the image scanner that captures images of the back of the media items. This may be null in Common.Status if the physical device has no back scanner or the capability to report the status of the back scanner is not supported by the device, otherwise the following values are possible: • `ok` - The back scanner is OK. • `fading` - The back scanner performance is degraded. • `inoperative` - The back scanner is inoperative. • `unknown` - Status of the back scanner cannot be determined with the device in its current state. default: null

Properties
check/mICRReader Specifies the status of the MICR code line reader. This may be null in Common.Status if the physical device has no MICR code line reader or the capability to report the status of the MICR code line reader is not supported by the device, otherwise the following values are possible: • `ok` - The MICR code line reader is OK. • `fading` - The MICR code line reader performance is degraded. • `inoperative` - The MICR code line reader is inoperative. • `unknown` - Status of the MICR code line reader cannot be determined with the device in its current state. default: null
check/stacker Supplies the state of the stacker (also known as an escrow). The stacker is where the media items are held while the application decides what to do with them. This may be null in Common.Status if the physical device has no stacker or the capability to report the status of the stacker is not supported by the device, otherwise the following values are possible: • `empty` - The stacker is empty. • `notEmpty` - The stacker is not empty. • `full` - The stacker is full. This state is set if the number of media items on the stacker has reached maxMediaOnStacker or some physical limit has been reached. • `inoperative` - The stacker is inoperative. • `unknown` - Due to a hardware error or other condition, the state of the stacker cannot be determined. default: null
check/rebuncher Supplies the state of the re-buncher (return stacker). The re-buncher is where media items are re-bunched ready for return to the customer. This may be null in Common.Status if the physical device has no re-buncher or the capability to report the status of the re-buncher is not supported by the device, otherwise the following values are possible: • `empty` - The re-buncher is empty. • `notEmpty` - The re-buncher is not empty. • `full` - The re-buncher is full. This state is set if the number of media items on the re-buncher has reached its physical limit. • `inoperative` - The re-buncher is inoperative. • `unknown` - Due to a hardware error or other condition, the state of the re-buncher cannot be determined. default: null
check/mediaFeeder Supplies the state of the media feeder. This value indicates if there are items on the media feeder waiting for processing via the Check.GetNextItem command. If null, the device has no media feeder or the capability to report the status of the media feeder is not supported by the device. This value can be one of the following values: • `empty` - The media feeder is empty. • `notEmpty` - The media feeder is not empty. • `inoperative` - The media feeder is inoperative. • `unknown` - Due to a hardware error or other condition, the state of the media feeder cannot be determined. default: null
check/positions Specifies the status of the input, output and refused positions. This may be null in Common.StatusChangedEvent if no position states have changed. Property value constraints: `minProperties: 1` default: null

Properties
check/positions/input Specifies the status of the input position. This may be null in Common.StatusChangedEvent if no states have changed for the position. default: null
check/positions/input/shutter Specifies the state of the shutter. This property is null in Common.Status if the physical device has no shutter or shutter state reporting is not supported, otherwise the following values are possible: • `closed` - The shutter is operational and is closed. • `open` - The shutter is operational and is open. • `jammed` - The shutter is jammed and is not operational. • `unknown` - Due to a hardware error or other condition, the state of the shutter cannot be determined. default: null
check/positions/input/positionStatus The status of the position. This property is null in Common.Status if the physical device is not capable of reporting whether or not items are at the position, otherwise 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. default: null
check/positions/input/transport Specifies the state of the transport mechanism. The transport is defined as any area leading to or from the position. This property is null in Common.Status if the physical device has no transport or transport state reporting is not supported, otherwise 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. default: null
check/positions/input/transportMediaStatus Returns information regarding items which may be present on the transport. This property is null in Common.Status if the physical device is not capable of reporting whether or not items are on the transport, otherwise the following values are possible: • `empty` - The transport is empty. • `notEmpty` - The transport is not empty. • `unknown` - Due to a hardware error or other condition it is not known whether there are items on the transport. default: null
check/positions/input/jammedShutterPosition Returns information regarding the position of the jammed shutter. This property is null in Common.Status if the physical device has no shutter or the reporting of the position of a jammed shutter is not supported, otherwise the following values are possible: • `notJammed` - The shutter is not jammed. • `open` - The shutter is jammed, but fully open. • `partiallyOpen` - The shutter is jammed, but partially open. • `closed` - The shutter is jammed, but fully closed. • `unknown` - The position of the shutter is unknown. default: null
check/positions/output Specifies the status of the output position. This may be null in Common.StatusChangedEvent if no states have changed for the position. default: null

Properties
check/positions/refused Specifies the status of the refused position. This may be null in Common.StatusChangedEvent if no states have changed for the position. default: null
mixedMedia Status information for XFS4IoT services implementing the MixedMedia interface. This will be null if the MixedMedia interface is not supported. default: null
mixedMedia/modes Specifies the state of the transaction modes supported by the Service. Property value constraints: `minProperties: 1`
mixedMedia/modes/cashAccept Specifies whether transactions can accept cash. This property may be null if no change required or its state has not changed in Common.StatusChangedEvent. default: null
mixedMedia/modes/checkAccept Specifies whether transactions can accept checks. This property may be null if no change required or its state has not changed in Common.StatusChangedEvent. default: null
keyManagement Status information for XFS4IoT services implementing the KeyManagement interface. This will be null if the KeyManagement interface is not supported. default: null
keyManagement/encryptionState Specifies the state of the encryption module. This may be null in Common.StatusChangedEvent if unchanged. default: null
keyManagement/certificateState Specifies the state of the public verification or encryption key in the PIN certificate modules. This may be null in Common.StatusChangedEvent if unchanged. default: null
keyboard Status information for XFS4IoT services implementing the Keyboard interface. This will be null if the Keyboard interface is not supported. default: null
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. This may be null in Common.StatusChangedEvent if unchanged. default: null
keyboard/autoBeepMode/inactiveAvailable Specifies whether an automatic tone will be generated for all inactive keys. This may be null in Common.StatusChangedEvent if unchanged. default: null

Properties
textTerminal Status information for XFS4IoT services implementing the TextTerminal interface. This will be null if the TextTerminal interface is not supported. default: null
textTerminal/keyboard Specifies the state of the keyboard in the text terminal unit. This property will be null in Common.Status if the keyboard is not available, otherwise one of the following values: • `on` - The keyboard is activated. • `off` - The keyboard is not activated. default: null
textTerminal/keyLock Specifies the state of the keyboard lock of the text terminal unit. This property will be null in Common.Status if the keyboard lock switch is not available, otherwise one of the following values: • `on` - The keyboard lock switch is activated. • `off` - The keyboard lock switch is not activated. default: null
textTerminal/displaySizeX Specifies the horizontal size of the display of the text terminal unit (the number of columns that can be displayed). This property will be null in Common.StatusChangedEvent if unchanged. Property value constraints: `minimum: 0` default: null
textTerminal/displaySizeY Specifies the vertical size of the display of the text terminal unit (the number of rows that can be displayed). This property will be null in Common.StatusChangedEvent if unchanged. Property value constraints: `minimum: 0` default: null
printer Status information for XFS4IoT services implementing the Printer interface. This will be null if the Printer interface is not supported. default: null
printer/media Specifies the state of the print media (i.e. receipt, statement, passbook, etc.) as one of the following values. This property will be null in Common.Status for journal printers or if 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. default: null

Properties
printer/paper Specifies the state of paper supplies as one of the following values. Each individual supply state will be null in Common.Status 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. default: null
printer/paper/upper The state of the upper paper supply. default: null
printer/paper/lower The state of the lower paper supply. default: null
printer/paper/external The state of the external paper supply. default: null
printer/paper/aux The state of the auxiliary paper supply. default: null
printer/paper/aux2 The state of the second auxiliary paper supply. default: null
printer/paper/park The state of the parking station paper supply. default: null
printer/paper/vendorSpecificPaperSupply (example name) The state of the additional vendor specific paper supplies. default: null
printer/toner Specifies the state of the toner or ink supply or the state of the ribbon. The property will be null in Common.Status if the capability is not supported by device, otherwise one of the following: • `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. default: null
printer/ink Specifies the status of the stamping ink in the printer. The property will be null in Common.Status if the capability is not supported by device, otherwise one of the following: • `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. default: null

Properties
printer/lamp Specifies the status of the printer imaging lamp. The property will be null in Common.Status if the capability is not supported by device, otherwise one of the following: • `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. default: null
printer/retractBins An array of bin state objects. If no retain bins are supported, the property will be null. default: null
printer/retractBins/state Specifies the state of the printer retract bin as one of the following. This may be null in Common.StatusChangedEvent if unchanged. • `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. default: null
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. This may be null in Common.StatusChangedEvent if unchanged. Property value constraints: `minimum: 0` default: null
printer/mediaOnStacker The number of media on stacker; applicable only to printers with stacking capability therefore null if not applicable. Property value constraints: `minimum: 0` default: null
printer/paperType Specifies the type of paper loaded as one of the following. Only applicable properties are reported. This may be null in Common.StatusChangedEvent if unchanged. • `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. default: null
printer/paperType/upper The upper paper supply paper type. default: null
printer/paperType/lower The lower paper supply paper type. default: null
printer/paperType/external The external paper supply paper type. default: null

Properties
printer/paperType/aux The auxililliary paper supply paper type. default: null
printer/paperType/aux2 The second auxililliary paper supply paper type. default: null
printer/paperType/park The parking station paper supply paper type. default: null
printer/paperType/exampleProperty1 (example name) The additional vendor specific paper types. default: null
printer/blackMarkMode Specifies the status of the black mark detection and associated functionality. The property is null if 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. default: null
barcodeReader Status information for XFS4IoT services implementing the Barcode Reader interface. This will be null if the Barcode Reader interface is not supported. default: null
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 null if the Biometrics interface is not supported. default: null
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). This property is null if the physical device does not support the ability to report whether or not a subject is on the scanning position. default: null
biometric/capture Indicates whether scanned biometric data has been captured using the Biometric.Read and is currently stored and ready for comparison. This will be set to false when scanned data is cleared using the Biometric.Clear. This may be null in Common.StatusChangedEvent if unchanged. default: null

Properties
biometric/dataPersistence Specifies the current data persistence mode. The data persistence mode controls how biometric data that has been captured using the Biometric.Read will be handled. This property is null if the property persistenceModes is null or both properties persist and clear are false. 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. default: null
biometric/remainingStorage Specifies how much of the reserved storage specified by the capability templateStorage is remaining for the storage of templates in bytes. if null, this property is not supported. Property value constraints: `minimum: 0` default: null
camera Status information for XFS4IoT services implementing the Camera interface. This will be null if the Camera interface is not supported. default: null
camera/media Specifies the state of the recording media of the cameras as one of the following. For a device which stores pictures on a hard disk drive or other general-purpose storage, the relevant property will be null. This property may be null in Common.StatusChangedEvent if unchanged. • `ok` - The media is in a good state. • `high` - The media is almost full (threshold). • `full` - The media is full. • `unknown` - Due to a hardware error or other condition, the state of the media cannot be determined. default: null
camera/media/room Specifies the state of the recording media of the camera that monitors the whole self-service area. default: null
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. default: null
camera/media/exitSlot Specifies the state of the recording media of the camera that monitors the exit slot(s) of the self-service machine. default: null
camera/media/vendorSpecificCameraMedia (example name) Allows vendor specific cameras to be reported. default: null
camera/cameras Specifies the state of the cameras as one of the following. The relevant property will be null if not supported and this property may be null in Common.StatusChangedEvent if unchanged. • `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. default: null

Properties
camera/cameras/room Specifies the state of the camera that monitors the whole self-service area. default: null
camera/cameras/person Specifies the state of the camera that monitors the person standing in front of the self-service machine. default: null
camera/cameras/exitSlot Specifies the state of the camera that monitors the exit slot(s) of the self-service machine. default: null
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 relevant camera's property is 0. Properties may be null in Common.StatusChangedEvent if unchanged. default: null
camera/pictures/room Specifies the number of pictures stored on the recording media of the room camera. Property value constraints: `minimum: 0` default: null
camera/pictures/person Specifies the number of pictures stored on the recording media of the person camera. Property value constraints: `minimum: 0` default: null
camera/pictures/exitSlot Specifies the number of pictures stored on the recording media of the exit slot camera. Property value constraints: `minimum: 0` default: null
camera/pictures/vendorSpecificCameraPictures (example name) Allows vendor specific cameras to be reported. Property value constraints: `minimum: 0` default: null
lights Status information for XFS4IoT services implementing the Lights interface. This will be null if the Lights interface is not supported. default: null
lights/cardReader Card Reader Light. This property is null if not applicable. default: null

Properties
lights/cardReader/position The light position. Can be used for devices which have multiple input and output positions. This may be 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. • `default` - The default position.
lights/cardReader/flashRate The light flash rate. This may be null in Common.StatusChangedEvent if unchanged, otherwise 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). default: null
lights/cardReader/color The light color. This may be null in Common.StatusChangedEvent if unchanged, otherwise 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. default: null
lights/cardReader/direction The light direction, The value can be null if not required. One of the following values: • `entry` - The light is indicating entry. • `exit` - The light is indicating exit. default: null
lights/pinPad Pin Pad Light. This property is null if not applicable. default: null
lights/notesDispenser Notes Dispenser Light. This property is null if not applicable. default: null
lights/coinDispenser Coin Dispenser Light. This property is null if not applicable. default: null
lights/receiptPrinter Receipt Printer Light. This property is null if not applicable. default: null

Properties
lights/passbookPrinter Passbook Printer Light. This property is null if not applicable. default: null
lights/envelopeDepository Envelope Depository Light. This property is null if not applicable. default: null
lights/checkUnit Check Unit Light. This property is null if not applicable. default: null
lights/billAcceptor Bill Acceptor Light. This property is null if not applicable. default: null
lights/envelopeDispenser Envelope Dispenser Light. This property is null if not applicable. default: null
lights/documentPrinter Document Printer Light. This property is null if not applicable. default: null
lights/coinAcceptor Coin Acceptor Light. This property is null if not applicable. default: null
lights/scanner Scanner Light. This property is null if not applicable. default: null
lights/contactless Contactless Reader Light. This property is null if not applicable. default: null
lights/cardReader2 Card Reader 2 Light. This property is null if not applicable. default: null
lights/notesDispenser2 Notes Dispenser 2 Light. This property is null if not applicable. default: null
lights/billAcceptor2 Bill Acceptor 2 Light. This property is null if not applicable. default: null
lights/statusGood Status Indicator light - Good. This property is null if not applicable. default: null
lights/statusWarning Status Indicator light - Warning. This property is null if not applicable. default: null
lights/statusBad Status Indicator light - Bad. This property is null if not applicable. default: null

Properties
lights/statusSupervisor Status Indicator light - Supervisor. This property is null if not applicable. default: null
lights/statusInService Status Indicator light - In Service. This property is null if not applicable. default: null
lights/fasciaLight Fascia Light. This property is null if not applicable. default: null
lights/vendorSpecificLight (example name) Additional vendor specific lights. default: null
auxiliaries Status information for XFS4IoT services implementing the Auxiliaries interface. This will be null if the Auxiliaries interface is not supported. default: null
auxiliaries/operatorSwitch Specifies the state of the Operator switch. • `run` - The switch is in run mode. • `maintenance` - The switch is in maintenance mode. • `supervisor` - The switch is in supervisor mode. This property is null if not applicable. default: null
auxiliaries/tamperSensor Specifies the state of the Tamper sensor. • `off` - There is no indication of a tampering attempt. • `on` - There has been a tampering attempt. This property is null if not applicable. default: null
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: • `off` - There is no indication of a tampering attempt. • `on` - There has been a tampering attempt. This property is null if not applicable. default: null
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: • `off` - The seismic activity has not been high enough to trigger the sensor. • `on` - The seismic or other activity has triggered the sensor. This property is null if not applicable. default: null

Properties
auxiliaries/heatSensor Specifies the state of the Heat Sensor. This sensor is triggered by excessive heat (fire) near the terminal. Specified as one of the following: • `off` - The heat has not been high enough to trigger the sensor. • `on` - The heat has been high enough to trigger the sensor. This property is null if not applicable. default: null
auxiliaries/proximitySensor Specifies the state of the Proximity Sensor. This sensor is triggered by movements around the terminal. Specified as one of the following: • `present` - The sensor is showing that there is someone present at the terminal. • `notPresent` - The sensor can not sense any people around the terminal. This property is null if not applicable. default: null
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: • `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. This property is null if not applicable. default: null
auxiliaries/enhancedAudioSensor Specifies the presence or absence of a consumer′s headphone connected to the Audio Jack. Specified as one of the following: • `present` - There is a headset connected. • `notPresent` - There is no headset connected. This property is null if not applicable. default: null
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: • `off` - The sensor has not been triggered. • `on` - The terminal is about to be rebooted or shutdown. This property is null if not applicable. default: null
auxiliaries/consumerDisplaySensor Specifies the state of the Consumer Display. Specified as one of the following: • `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. This property is null if not applicable. default: null

Properties
auxiliaries/operatorCallButtonSensor Specifies the state of the Operator Call Button as one of the following: • `off` - The Operator Call Button is released (not pressed). • `on` - The Operator Call Button is being pressed. This property is null if not applicable. default: null
auxiliaries/handsetSensor Specifies the state of the Handset, which is a device similar to a telephone receiver. Specified as one of the following: • `onTheHook` - The Handset is on the hook. • `offTheHook` - The Handset is off the hook. This property is null if not applicable. default: null
auxiliaries/headsetMicrophoneSensor Specifies the presence or absence of a consumer′s headset microphone connected to the Microphone Jack. Specified as one of the following: • `present` - There is a headset microphone connected. • `notPresent` - There is no headset microphone connected. This property is null if not applicable. default: null
auxiliaries/fasciaMicrophoneSensor Specifies the state of the fascia microphone as one of the following: • `off` - The Fascia Microphone is turned off. • `on` - The Fascia Microphone is turned on. This property is null if not applicable. default: null
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: • `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. This property is null if not applicable. default: null
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: • `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. This property is null if not applicable. default: null

Properties
auxiliaries/cabinetFrontDoor Specifies the overall state of the Front Cabinet Doors. The front is defined as the side facing the customer/consumer. Cabinet Doors are doors that open up for consumables, and hardware that does not have to be in a secure place. Specified as one of the following: • `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. This property is null if not applicable. default: null
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: • `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. This property is null if not applicable. default: null
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: • `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. This property is null if not applicable. default: null
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: • `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. This property is null if not applicable. default: null
auxiliaries/openClosedIndicator Specifies the state of the Open/Closed Indicator as one of the following: • `closed` - The terminal is closed for a consumer. • `open` - The terminal is open to be used by a consumer. This property is null if not applicable. default: null

Properties
auxiliaries/audio Specifies the state of the Audio Indicator. This property is null if not applicable. default: null
auxiliaries/audio/rate Specifies the state of the Audio Indicator as one of the following values. This may be null in Common.StatusChangedEvent if unchanged. • `on` - Turn on the Audio Indicator. • `off` - Turn off the Audio Indicator. • `continuous` - Turn the Audio Indicator to continuous. This property is null if not applicable. default: null
auxiliaries/audio/signal Specifies the Audio sound as one of the following values. This may be null in Common.StatusChangedEvent if unchanged. • `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. This property is null if not applicable. default: null
auxiliaries/heating Specifies the state of the internal heating as one of the following: • `off` - The internal heating is turned off. • `on` - The internal heating is turned on. This property is null if not applicable. default: null
auxiliaries/consumerDisplayBacklight Specifies the Consumer Display Backlight as one of the following: • `off` - The Consumer Display Backlight is turned off. • `on` - Consumer Display Backlight is turned on. This property is null if not applicable. default: null
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: • `off` - The Signage Display is turned off. • `on` - The Signage Display is turned on. This property is null if not applicable. default: null

Properties

auxiliaries/volume
Specifies the value of the Volume Control. 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. This property is null if not applicable.
Property value constraints:
`minimum: 1`
`maximum: 1000`
default: null

auxiliaries/UPS Specifies the state of the Uninterruptible Power Supply. This property is null if not applicable. Properties contained in this property may be null in Common.StatusChangedEvent if unchanged. default: null
auxiliaries/UPS/low The charge level of the UPS is low. default: null
auxiliaries/UPS/engaged The UPS is engaged. default: null
auxiliaries/UPS/powering The UPS is powering the system. default: null
auxiliaries/UPS/recovered The UPS was engaged when the main power went off. default: null
auxiliaries/audibleAlarm Species the state of the Audible Alarm device as one of the following: • `off` - The Alarm is turned off. • `on` - The Alarm is turned on. This property is null if not applicable. default: null

Properties

auxiliaries/enhancedAudioControl

Specifies the state of the Enhanced Audio Controller. The Enhanced Audio Controller controls how private and public audio are broadcast when the headset is inserted into/removed from the audio jack and when the handset is off-hook/on-hook. In the following, Privacy Device is used to refer to either the headset or handset. The Enhanced Audio Controller state is specified as one of the following:

- `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.

This property is null if not applicable.

default: null

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 following values:

- `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.

This property is null if not applicable.

default: null

auxiliaries/microphoneVolume

Specifies the value of the Microphone Volume Control. 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. This property is null if not applicable.

Property value constraints:

```
minimum: 1
maximum: 1000
```

default: null

vendorMode

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

default: null

vendorMode/device

Specifies the status of the Vendor Mode Service. This property may be null in events if the status did not change, otherwise will be one of the following values:

- `online` - The Vendor Mode service is available.
- `offline` - The Vendor Mode service is not available.

default: null

Properties
vendorMode/service Specifies the service state. This property may be null in events if the state did not change, otherwise will be 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. default: null
vendorApplication Status information for XFS4IoT services implementing the Vendor Application interface. This will be null in Common.Status if the interface is not supported. default: null
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.

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 2.0)	Type	Required
{		
"eventId": "hardware",	string	✓
"action": "reset",	string, null	
"vendorDescription": "An error occurred in position B."	string, null	
}		

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 null 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*.

default: null

vendorDescription

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

default: null

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 2.0)	Type	Required
{		
"reasonDescription": "Nonce cleared by timeout"	string, null	
}		

Properties

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

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
cardreader-5	ISO 4217

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

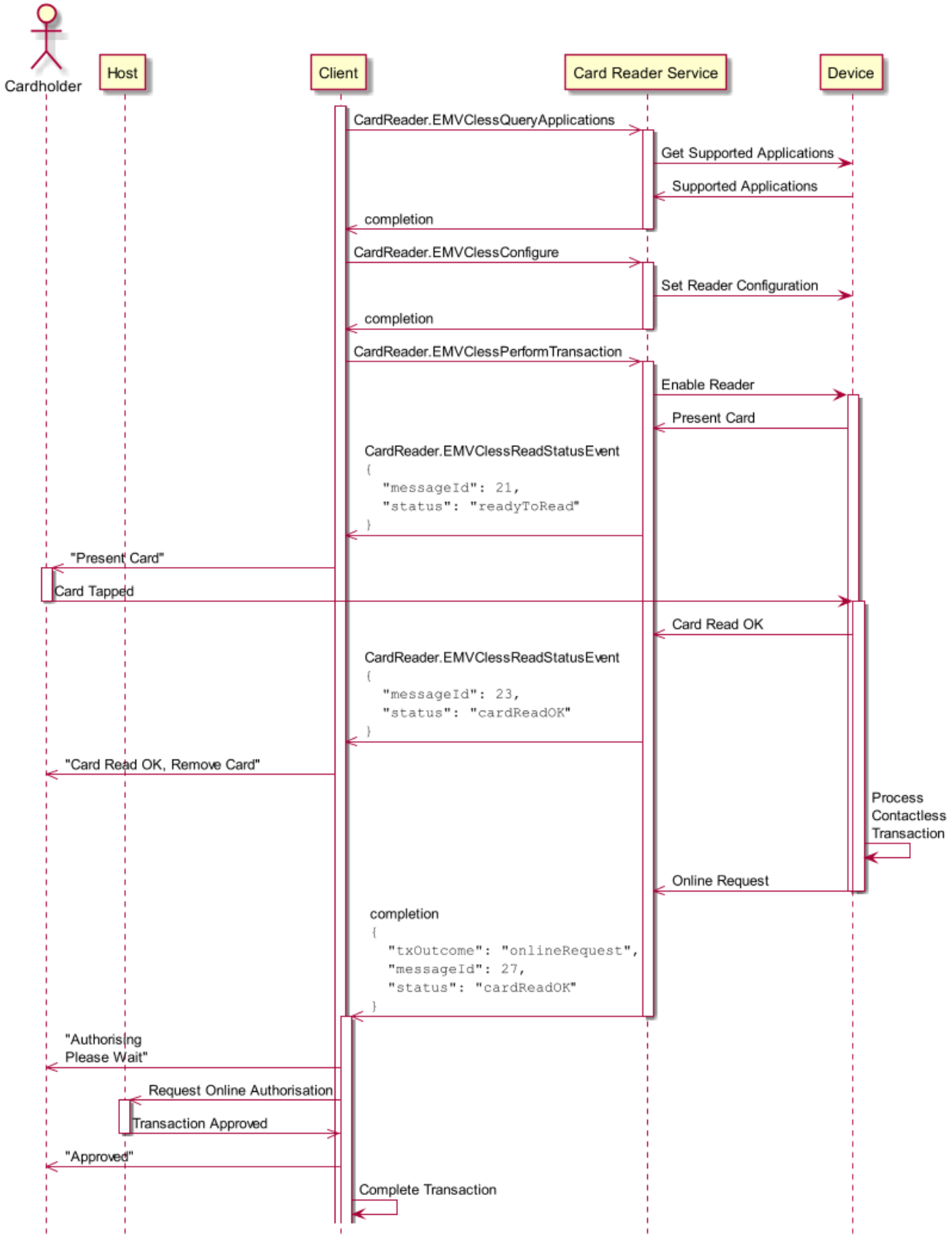
and prepare the intelligent contactless card reader for a transaction and are also part of the data that is returned by the reader on completion of a card tap.

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.

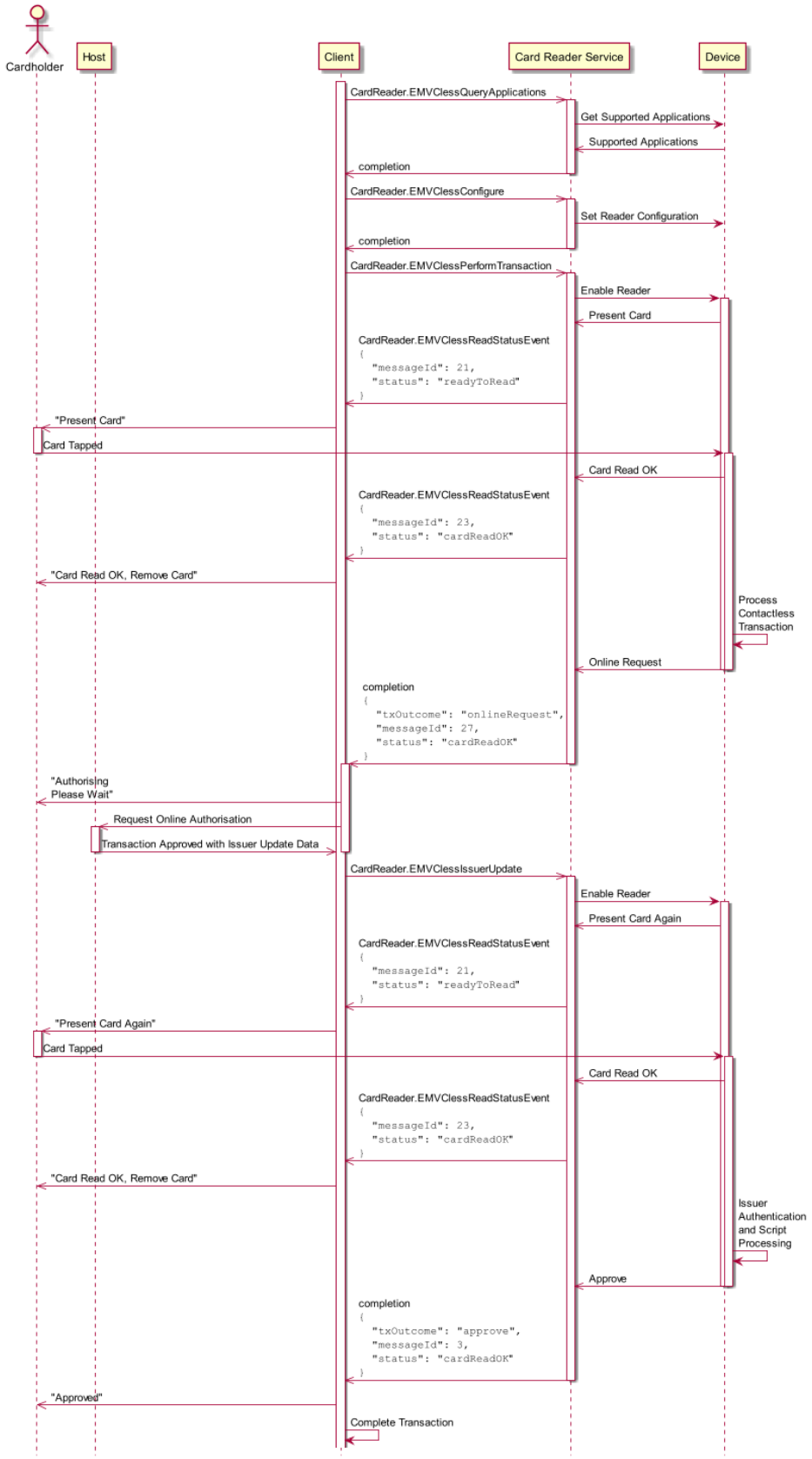
5.1.3 Intelligent Contactless Card Reader Sequence Diagrams

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

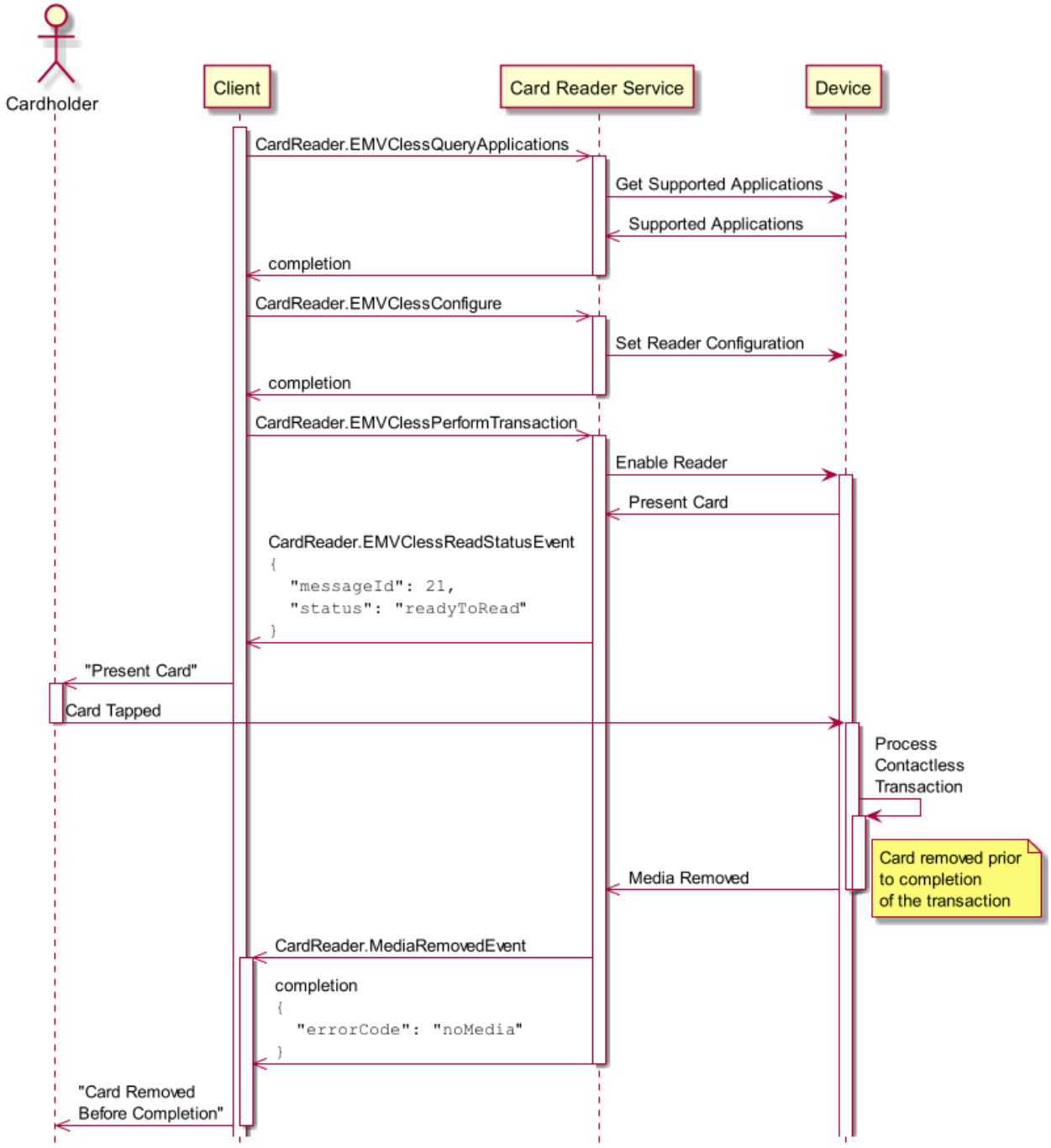
Single Tap Transaction Without Issuer Update Processing



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 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
{		
"ifmIdentifiers": {	object, null	
"*emv*": "Example IFM Identifier",	string	
"*europay*": See *ifmIdentifiers/emv*	string	
}		
}		

Properties

ifmIdentifiers

An array of the IFM identifiers supported by the Service or null if none are supported.

default: null

ifmIdentifiers/emv (example name)

Specifies a single IFM identifier supported by the Service. The property name is the IFM authority, the property value is the 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 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"appData": [{`	array (object), null	
`"aid": "oAAAAAMQEA==",`	string	✓
`"kernelIdentifier": "Ag=="`	string, null	
`}]`		
`}`		

Properties

appData

An array of application data objects which specifies a supported application identifier (AID) and associated Kernel Identifier.

Property value constraints:

```
minItems: 1
```

default: null

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 null 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
```

default: null

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.

For non-motorized Card Readers which read track data on card exit, the *invalidData* completion 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 2.0)	Type	Required
{		
"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

track1
Track 1 of the magnetic stripe will be read.
default: false

track2
Track 2 of the magnetic stripe will be read.
default: false

Properties
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 2.0)	Type	Required
{		
"errorCode": "mediaJam",	string, null	
"track1": {	object, null	

Payload (version 2.0)	Type	Required
"status": "dataMissing",	string, null	
"data": "QmFzZTY0IGVuY29kZWQg ..."	string, null	
},		
"track2": See track1 properties	object, null	
"track3": See track1 properties	object, null	
"chip": [{	array (object), null	
"status": See track1/status,	string, null	
"data": See track1/data	string, null	
}],		
"security": {	object, null	
"status": See track1/status,	string, null	
"data": "readLevel1"	string, null	
},		
"watermark": See track1 properties	object, null	
"memoryChip": {	object, null	
"status": See track1/status,	string, null	
"protocol": "chipT0",	string	✓
"data": "O2gAUACFyEARAJAC"	string, null	
},		
"track1Front": See track1 properties	object, null	
"frontImage": "wCAAAQgwMDAwMDAwMA==",	string, null	
"backImage": "wCAAAQgwMDAwMDAwMA==",	string, null	
"track1JIS": See track1 properties	object, null	
"track3JIS": See track1 properties	object, null	
"ddi": See track1 properties	object, null	
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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 card's security and no other data source was requested.
- `cardCollision` - There was an unresolved collision of two or more contactless card signals.

default: null

Properties
track1 Contains the data read from track 1. The property is null if not requested. default: null
track1/status The status values applicable to all data sources. This property is null if the data is OK. Possible values are: • `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. default: null

track1/data
Base64 encoded representation of the data. This property is null if not read.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`
default: null

track2 Contains the data read from track 2. The property is null if not requested. default: null
track3 Contains the data read from track 3. The property is null if not requested. default: null

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. The property is null if not requested.
Property value constraints:
`minProperties: 1`
`minItems: 1`
default: null

security Contains the data returned by the security module (i.e. MM, CIM86). If the check could not be executed, *status* and *data* indicate the cause of the security check failure; the *errorCode* will only be *securityFail* if no other data source is requested. The property is null if not requested. default: null

Properties
security/data The security data can be one of the following: This property is null if there is no security data on the card. • `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. • `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). default: null
watermark Contains the data read from the Swedish Watermark track. The property is null if not requested. default: null
memoryChip Memory Card Identification data read from the memory chip. The property is null if not requested. default: null
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. This property is null if not read. default: null
track1Front Contains the data read from the front track 1. In some countries this track is known as JIS II track. This property is null if not read. default: null

frontImage
Base64 encoded representation of the BMP image file for the front of the card.
The property is null if not requested or not read.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`
default: null

Properties

backImage
Base64 encoded representation of the BMP image file for the back of the card.
The property is null if not requested or not read.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`
default: null

track1JIS Contains the data read from JIS I track 1 (8bits/char). The property is null if not requested. default: null
track3JIS Contains the data read from JIS I track 3 (8bits/char). The property is null if not requested. default: null
ddi Contains the dynamic digital identification data read from magnetic stripe. The property is null if not requested. default: null

Event Messages

- CardReader.InsertCardEvent
- CardReader.MediaInsertedEvent
- 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 2.0)	Type	Required
{		
"track1": {	object, null	
"data": "QmFzZTY0IGVuY29kZWQg ...",	string	✓
"writeMethod": "loco"	string, null	
},		
"track2": See track1 properties	object, null	
"track3": See track1 properties	object, null	
"track1Front": See track1 properties	object, null	
"track1JIS": See track1 properties	object, null	
"track3JIS": See track1 properties	object, null	
"additonalProperties": See track1 properties	object, null	
}		

Properties

track1

Specifies data is to be written to track 1. This property is null if not applicable.

default: null

track1/data

Base64 encoded representation of the data

Property value constraints:

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

track1/writeMethod

Indicates whether a low coercivity or high coercivity magnetic stripe is to be written. If this property is null, the service will determine whether low or high coercivity is to be used.

Specifies as one of the following:

- `loco` - Write using low coercivity.
- `hico` - Write using high coercivity.

default: null

Properties
track2 Specifies data is to be written to track 2. This property is null if not applicable. default: null
track3 Specifies data is to be written to track 3. This property is null if not applicable. default: null
track1Front Specifies data is to be written to the front track 1. In some countries this track is known as JIS II track. This property is null if not applicable. default: null
track1JIS Specifies data is to be written to JIS I track 1 (8bits/char). This property is null if not applicable. default: null
track3JIS Specifies data is to be written to JIS I track 3 (8bits/char). This property is null if not applicable. default: null
additonalProperties Specifies data is to be written to vendor specific track. This property is null if not applicable. default: null

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "mediaJam"`	string, null	
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null

Event Messages

- CardReader.InsertCardEvent
- CardReader.MediaInsertedEvent
- 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 2.0)	Type	Required
`{`		
`"from": "unit1",`	string	
`"to": "exit"`	string	
`}`		

Properties

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 2.0)	Type	Required
`{`		
`"errorCode": "mediaJam"`	string, null	
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null

Event Messages

None

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 2.0)	Type	Required
{		
"keyValue": "QmFzZTY0IGVuY29kZWQg ..."	string	✓
}		

Properties

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

Payload (version 2.0)	Type	Required
{		
"errorCode": "invalidKey"	string, null	
}		

Properties

errorCode

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

- `invalidKey` - The key does not fit to the security module.

default: null

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 2.0)	Type	Required
{		
"chipProtocol": "chipT1",	string	
"chipData": "wCAAAQgwMDAwMDAwMA=="	string	✓
}		

Properties

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.

default: "chipProtocolNotRequired"

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 2.0)	Type	Required
{		
"errorCode": "mediaJam",	string, null	
"chipProtocol": "chipT0",	string, null	

Payload (version 2.0)	Type	Required
"chipData": "bGs="	string, null	
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

chipProtocol

Identifies the protocol that is used to communicate with the chip. This contains the same value as the corresponding property in the payload. This property is null for Memory Card dialogs.

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.

default: null

chipData

The Base64 encoded data received from the chip. This property is null if no data received.

Property value constraints:

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

default: null

Event Messages

None

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 2.0)	Type	Required
{		
"to": "retain",	string	
"storageId": "unit4"	string, null	
}		

Properties

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.
- `auto` - 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.

default: "auto"

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 null, 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]+$
```

default: null

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "mediaJam"	string, null	
}		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null

Event Messages

None

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 2.0)	Type	Required
{		
"chipPower": "cold"	string	✓
}		

Properties

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 2.0)	Type	Required
{		
"errorCode": "chipPowerNotSupported",	string, null	
"chipData": "O2gAUACFyEARAJAC"	string, null	
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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).

default: null

chipData

The Base64 encoded data received from the chip. This property is null if no data received.

Property value constraints:

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

default: null

Event Messages

None

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 2.0)	Type	Required
{		
"terminalData": "wCAAAQgwMDAwMDAwMA==",	string	✓
"aidData": [{	array (object)	✓
"aid": "oAAAAAMQEA==",	string	✓
"partialSelection": false,	boolean	
"transactionType": 0,	integer	✓
"kernelIdentifier": "Ag==",	string, null	
"configData": "nwYHoAAAASFHEQ=="	string	✓
}],		
"keyData": [{	array (object), null	
"rid": "oAAAAAM=",	string	✓
"caPublicKey": {	object	✓
"index": 0,	integer	✓
"algorithmIndicator": 0,	integer	✓
"exponent": "AQAB",	string	✓
"modulus": "Kjyq8qcAWnJB66p3cREs ...",	string	✓
"checksum": "7hURzscQIKm5BEOzex1f ..."	string	✓
}		
}]		
}		

Properties

terminalData

Base64 encoded representation of the BER-TLV formatted data for the terminal e.g. Terminal Type, Transaction Category Code, Merchant Name & Location etc. Any terminal based data elements referenced in the Payment Systems Specifications or EMVCo Contactless Payment Systems Specifications Books may be included (see [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. default: false
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]. Property value constraints: `minimum: 0`

aidData/kernelIdentifier
Base64 encoded representation of the EMVCo defined kernel identifier associated with the *aid*. This will be ignored if the reader does not support kernel identifiers. This property is null if not applicable.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`
default: null

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. This property is null if not applicable. default: null
keyData/rid Specifies the payment system's Registered Identifier (RID). RID is the first 5 bytes of the AID and identifies the payments system. Property value constraints: `format: base64`
keyData/caPublicKey CA Public Key information for the specified *rid*.
keyData/caPublicKey/index Specifies the CA Public Key Index for the specific *rid*. Property value constraints: `minimum: 0`

Properties
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. Property value constraints: `minimum: 0`

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 2.0)	Type	Required
`{`		
`"errorCode": "invalidTerminalData"`	string, null	
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null

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 2.0)	Type	Required
{		
"data": "XyoCCXiaAxcICJwBAJ8C ..."	string, null	
}		

Properties

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 property can therefore be null.

*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
```

default: null

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "noMedia",	string, null	
"chip": {	object, null	
"txOutcome": "multipleCards",	string	✓
"cardholderAction": "none",	string	✓
"dataRead": "fSfILqum6niI6jURWzeo ...",	string	✓

Payload (version 2.0)	Type	Required
"clessOutcome": {	object, null	
"cvm": "onlinePIN",	string	✓
"alternateInterface": "magneticStripe",	string, null	
"receipt": false,	boolean	
"uiOutcome": {	object, null	
"messageId": 0,	integer	✓
"status": "notReady",	string	✓
"holdTime": 0,	integer	
"valueDetails": {	object, null	
"qualifier": "amount",	string	✓
"value": "000000012345",	string	✓
"currencyCode": 826	integer	✓
},		
"languagePreferenceData": "en"	string, null	
},		
"uiRestart": See chip/clessOutcome/uiOutcome properties	object, null	
"fieldOffHoldTime": 0,	integer	
"cardRemovalTimeout": 0,	integer	
"discretionaryData": "Qnl0ZSBBcnJheSBEYXRh"	string, null	
}		
},		
"track1": See chip properties	object, null	
"track2": See chip properties	object, null	
"track3": See chip properties	object, null	
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

chip

Contains the BER-TLV formatted data read from the chip. This property is set after the contactless transaction has been completed with EMV mode or mag-stripe mode. This property is null if not applicable.

default: null

Properties

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).

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 null for contactless chip card readers that do not follow EMVCo Entry Point Specifications.

default: null

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.

Properties
chip/clessOutcome/alternateInterface This specifies the alternative interface to be used to complete a transaction if applicable as one of the following: • `contact` - *txOutcome* is *tryAnotherInterface* and the contact chip interface should be used to complete a transaction. • `magneticStripe` - *txOutcome* is *tryAnotherInterface* and the magnetic stripe interface should be used to complete a transaction. • `null` - *txOutcome* is not *tryAnotherInterface* default: null
chip/clessOutcome/receipt Specifies whether a receipt should be printed. default: false
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 null. 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. default: null
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). Property value constraints: `minimum: 0`
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
chip/clessOutcome/uiOutcome/valueDetails Indicates a value associated with a transaction, either an amount or a balance. See [Ref. cardreader-3] for more details. This property will be null if no amount or balance is applicable. default: null
chip/clessOutcome/uiOutcome/valueDetails/qualifier Qualifies *value*. This data is defined by EMVCo as one of the following: • `amount` - *value* is an Amount. • `balance` - *value* is a Balance.

Properties
chip/clessOutcome/uiOutcome/valueDetails/value Represents the numeric value of the amount or balance (as specified by *qualifier*) to be displayed where appropriate. The format of this property is defined by EMVCo. Property value constraints: `pattern: ^[0-9]{12}$`

chip/clessOutcome/uiOutcome/valueDetails/currencyCode
Represents the numeric value of the currency code as defined by ISO 4217 [Ref. cardreader-5].
Property value constraints:
`minimum: 0`
`maximum: 999`

chip/clessOutcome/uiOutcome/languagePreferenceData Represents the language preference (EMV Tag '5F2D') if returned by the card. If not returned, this property reports null. 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}` default: null
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 null. default: null
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` default: 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 null.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]*={0,2}$`
`format: base64`
default: null

track1 Contains the chip returned data formatted in as track 1. This property is set after the contactless transaction has been completed with mag-stripe mode. This property is null if not applicable. default: null
track2 Contains the chip returned data formatted in as track 2. This property is set after the contactless transaction has been completed with mag-stripe mode. This property is null if not applicable. default: null

Properties
track3 Contains the chip returned data formatted in as track 3. This property is set after the contactless transaction has been completed with mag-stripe mode. This property is null if not applicable. default: null

Event Messages

- CardReader.EMVClessReadStatusEvent

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 2.0)	Type	Required
{		
"data": "XyoCCXiaAxcICJwBAJ8C ..."	string	✓
}		

Properties

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 2.0)	Type	Required
{		
"errorCode": "noMedia",	string, null	
"chip": {	object, null	
"txOutcome": "multipleCards",	string	✓
"dataRead": "fSfILqum6niI6jURWzeo ...",	string	✓
"clessOutcome": {	object, null	
"cvm": "onlinePIN",	string	✓
"alternateInterface": "magneticStripe",	string, null	
"receipt": false,	boolean	
"uiOutcome": {	object, null	
"messageId": 0,	integer	✓
"status": "notReady",	string	✓
"holdTime": 0,	integer	
"valueDetails": {	object, null	
"qualifier": "amount",	string	✓

Payload (version 2.0)	Type	Required
"value": "000000012345",	string	✓
"currencyCode": 826	integer	✓
},		
"languagePreferenceData": "en"	string, null	
},		
"uiRestart": See chip/clessOutcome/uiOutcome properties	object, null	
"fieldOffHoldTime": 0,	integer	
"cardRemovalTimeout": 0,	integer	
"discretionaryData": "Qnl0ZSBBcnJheSBEYXRh"	string, null	
}		
}		
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

chip

Contains the BER-TLV formatted data read from the chip. This will be null if no data has been returned.

default: null

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].*

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.

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 null for contactless chip card readers that do not follow EMVCo Entry Point Specifications. default: null
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 This specifies the alternative interface to be used to complete a transaction if applicable as one of the following: • `contact` - *txOutcome* is *tryAnotherInterface* and the contact chip interface should be used to complete a transaction. • `magneticStripe` - *txOutcome* is *tryAnotherInterface* and the magnetic stripe interface should be used to complete a transaction. • `null` - *txOutcome* is not *tryAnotherInterface* default: null
chip/clessOutcome/receipt Specifies whether a receipt should be printed. default: false
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 null. 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. default: null
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). Property value constraints: `minimum: 0`
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/valueDetails Indicates a value associated with a transaction, either an amount or a balance. See [Ref. cardreader-3] for more details. This property will be null if no amount or balance is applicable. default: null
chip/clessOutcome/uiOutcome/valueDetails/qualifier Qualifies *value*. This data is defined by EMVCo as one of the following: • `amount` - *value* is an Amount. • `balance` - *value* is a Balance.
chip/clessOutcome/uiOutcome/valueDetails/value Represents the numeric value of the amount or balance (as specified by *qualifier*) to be displayed where appropriate. The format of this property is defined by EMVCo. Property value constraints: `pattern: ^[0-9]{12}$`

chip/clessOutcome/uiOutcome/valueDetails/currencyCode
Represents the numeric value of the currency code as defined by ISO 4217 [Ref. cardreader-5].
Property value constraints:
`minimum: 0`
`maximum: 999`

chip/clessOutcome/uiOutcome/languagePreferenceData Represents the language preference (EMV Tag '5F2D') if returned by the card. If not returned, this property reports null. 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}` default: null
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 null. default: null
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` default: 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

Properties

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

Event Messages

- CardReader.EMVClessReadStatusEvent

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 2.0)
This message does not define any properties.

5.3.2 CardReader.MediaInsertedEvent

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

Event Message

Payload (version 2.0)
`This message does not define any properties.`

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 2.0)
This message does not define any properties.

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 2.0)	Type	Required
{		
"track1": true,	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.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 2.0)	Type	Required
{		
"messageId": 0,	integer	✓
"status": "notReady",	string	✓
"holdTime": 0,	integer	
"valueDetails": {	object, null	
"qualifier": "amount",	string	✓
"value": "000000012345",	string	✓
"currencyCode": 826	integer	✓
},		
"languagePreferenceData": "en"	string, null	
}		

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).

Property value constraints:

```
minimum: 0
```

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

Properties
valueDetails Indicates a value associated with a transaction, either an amount or a balance. See [Ref. cardreader-3] for more details. This property will be null if no amount or balance is applicable. default: null
valueDetails/qualifier Qualifies *value*. This data is defined by EMVCo as one of the following: • `amount` - *value* is an Amount. • `balance` - *value* is a Balance.
valueDetails/value Represents the numeric value of the amount or balance (as specified by *qualifier*) to be displayed where appropriate. The format of this property is defined by EMVCo. Property value constraints: `pattern: ^[0-9]{12}$`

valueDetails/currencyCode
Represents the numeric value of the currency code as defined by ISO 4217 [Ref. cardreader-5].
Property value constraints:
`minimum: 0`
`maximum: 999`

languagePreferenceData Represents the language preference (EMV Tag '5F2D') if returned by the card. If not returned, this property reports null. 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}` default: null

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 2.0)
`This message does not define any properties.`

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

5.4.3 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.

Unsolicited Message

Payload (version 2.0)	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$|^unit[0-9A-Za-z]+$
```

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 - see Common.Capabilities classifications. 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 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"items": {`	object, null	
`"type20USD1": {`	object	
`"cashItem": {`	object	✓
`"noteID": 25,`	integer	✓
`"currency": "USD",`	string	✓
`"value": 20.00,`	number	✓
`"release": 1`	integer	
`},`		
`"enabled": true`	boolean	
`},`		
`"type10GBP2": See items/type20USD1 properties`	object	
`}`		
`}`		

Properties
items An object listing which cash items the device is capable of handling and whether the cash items are enabled for acceptance. May be null if empty. default: null
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`

Properties
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` default: 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 2.0)	Type	Required
{		
"tellerID": 0,	integer, null	
"currency": "USD"	string, null	
}		

Properties

tellerID
Identification of the teller. If invalid the error *invalidTellerId* is reported. If null, all tellers are reported.
Property value constraints:
`minimum: 0`
default: null

currency
ISO 4217 format currency identifier [Ref. cashmanagement-1]. If null, all currencies are reported for *tellerID*.
Property value constraints:
`pattern: ^[A-Z]{3}$`
default: null

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "invalidCurrency",	string, null	
"tellerDetails": [{	array (object), null	
"tellerID": 104,	integer	✓
"inputPosition": "inDefault",	string	
"outputPosition": "outDefault",	string	
"tellerTotals": {	object	✓
"*EUR*": {	object	
"itemsReceived": 1405.00,	number	
"itemsDispensed": 1405.00,	number	
"coinsReceived": 0.05,	number	
"coinsDispensed": 0.05,	number	
"cashBoxReceived": 1407.15,	number	
"cashBoxDispensed": 1407.15	number	
},		
"*GBP*": See *tellerDetails/tellerTotals/EUR* properties	object	
}		

Payload (version 2.0)	Type	Required
`}]`		
`}`		

Properties

errorCode

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

- `invalidCurrency` - Specified currency not currently available.
- `invalidTellerId` - Invalid teller ID.

default: null

tellerDetails

Array of teller detail objects. May be null if no teller defined.

default: null

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"

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}$
```

Properties
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` default: 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` default: 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` default: 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` default: 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` default: 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` default: 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 2.0)	Type	Required
{		
"action": "create",	string	✓
"tellerDetails": {	object	✓
"tellerID": 104,	integer	✓
"inputPosition": "inDefault",	string	
"outputPosition": "outDefault",	string	
"tellerTotals": {	object	✓
"*EUR*": {	object	
"itemsReceived": 1405.00,	number	
"itemsDispensed": 1405.00,	number	
"coinsReceived": 0.05,	number	
"coinsDispensed": 0.05,	number	
"cashBoxReceived": 1407.15,	number	
"cashBoxDispensed": 1407.15	number	
},		
"*GBP*": See *tellerDetails/tellerTotals/EUR* properties	object	
}		
}		
}		

Properties

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` default: 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` default: 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` default: 0

Properties
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` default: 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` default: 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` default: 0

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "invalidCurrency"`	string, null	
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null

Event Messages

None

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 2.0)	Type	Required
{		
"items": {	object, null	
"level": "fit",	string	✓
"index": 1	integer, null	
},		
"itemInfoType": {	object, null	
"serialNumber": true,	boolean	
"signature": true,	boolean	
"image": true	boolean	
}		
}		

Properties

items
Specifies which item or items to return information for. If null, all information on all items is returned.
default: null

Properties
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 null, all items of the specified *level* will be returned. Property value constraints: `minimum: 0` default: null
itemInfoType Specifies the type of information required. If null, all available information will be returned. default: null
itemInfoType/serialNumber Request the serial number of the item. default: true
itemInfoType/signature Request the signature of the item. default: true
itemInfoType/image Request the image of the item. default: true

Completion Message

Payload (version 2.0)	Type	Required
{		
"itemsList": [{	array (object), null	
"noteType": "type20USD1",	string, null	
"orientation": "frontTop",	string, null	
"signature": "MAA5ADgANwA2ADUANAAz ...",	string, null	
"level": "fit",	string	✓
"serialNumber": "AB12345YG",	string, null	
"image": "MAA5ADgANwA2ADUANAAz ...",	string, null	
"onClassificationList": "onClassificationList",	string, null	
"itemLocation": "unit1"	string	
}]		
}		

Properties
itemsList Array of objects listing the item information. May be null, if empty. default: null

Properties
itemsList/noteType A cash item as reported by CashManagement.GetBankNoteTypes. This is null if the item was not identified as a cash item. Property value constraints: `pattern: ^type[0-9A-Z]+$` default: null
itemsList/orientation Specifies the note orientation. This property is null if the hardware is not capable to determine the 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. default: null

itemsList/signature
Base64 encoded vendor specific signature data. If no signature is available or has not been requested then this is null.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`
default: null

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 null. default: null

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

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 null. 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. default: null
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]+$` default: "unknown"

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 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"version": "Version 1.2",`	string, null	
`"classificationElements": [{`	array (object), null	
`"serialNumber": "AB1234D",`	string	✓
`"currency": "USD",`	string	✓
`"value": 20.00,`	number	✓
`"level": "fit"`	string	✓
`}]`		
`}`		

Properties

version
This is an application defined string that sets the version identifier of the classification list. This property can be null if it has no version identifier.
default: null

classificationElements
Array of objects defining the classification list. May be null if empty.
default: null

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.

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 2.0)	Type	Required
`{`		
`"version": "Version 1.2",`	string, null	
`"classificationElements": [{`	array (object)	✓
`"serialNumber": "AB1234D",`	string	✓
`"currency": "USD",`	string	✓
`"value": 20.00,`	number	✓
`"level": "fit"`	string	✓
`}]`		
`}`		

Properties

version

This is an application defined string that sets the version identifier of the classification list. This property can be null if it has no version identifier.

default: null

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 2.0)
`This message does not define any properties.`

Event Messages

None

6.2.7 CashManagement.CloseShutter

This command closes the shutter.

Command Message

Payload (version 2.0)	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.

default: "outDefault"

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "unsupportedPosition"	string, null	
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

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 2.0)	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.

default: "outDefault"

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "unsupportedPosition"	string, null	
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

Event Messages

None

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 2.0)	Type	Required
{		
"location": {	object, null	
"outputPosition": "outDefault",	string	
"retractArea": "retract",	string	✓
"index": 1	integer, null	
}		
}		

Properties

location

Specifies where items are to be retracted from and where they are to be retracted to.

default: null

location/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"

location/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
location/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 and may be null. Property value constraints: `minimum: 1` default: null

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "cashUnitError",	string, null	
"storage": {	object, null	
"*unit1*": {	object, null	
"retractOperations": 15,	integer, null	
"deposited": {	object, null	
"unrecognized": 5,	integer, null	
"*type20USD1*": {	object, null	
"fit": 15,	integer, null	
"unfit": 0,	integer, null	
"suspect": 0,	integer, null	
"counterfeit": 0,	integer, null	
"inked": 0	integer, null	
},		
"*type50USD1*": See *storage/unit1/deposited/type20USD1* properties	object, null	
},		
"retracted": See storage/unit1/deposited properties	object, null	
"rejected": See storage/unit1/deposited properties	object, null	
"distributed": See storage/unit1/deposited properties	object, null	

Payload (version 2.0)	Type	Required
"transport": See storage/unit1/deposited properties	object, null	
},		
"unit2": See storage/unit1 properties	object, null	
},		
"transport": See storage/unit1/deposited properties	object, null	
"stacker": See storage/unit1/deposited properties	object, null	
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.
- `positionNotEmpty` - The retract area specified in *retractArea* is not empty so the retract operation is not possible.
- `incompleteRetract` - Some or all of the items were not retracted for a reason not covered by other error codes. The detail will be reported with the Dispenser.IncompleteRetractEvent.

default: null

storage

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.

default: null

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]+$
```

default: null

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. May be null in command data and events if not changing.

Property value constraints:

```
minimum: 0
```

default: null

Properties
storage/unit1/deposited The items deposited in the storage unit during a Cash In transaction. Can be null, if all values are 0. default: null
storage/unit1/deposited/unrecognized Count of unrecognized items handled by the cash interface. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/deposited/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification. default: null
storage/unit1/deposited/type20USD1/fit Count of genuine cash items which are fit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/deposited/type20USD1/unfit Count of genuine cash items which are unfit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/deposited/type20USD1/suspect Count of suspected counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/deposited/type20USD1/counterfeit Count of counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/deposited/type20USD1/inked Count of cash items which have been identified as ink stained. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
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*. Can be null, if all values are 0. default: null

Properties
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. Can be null, if all values are 0. default: null
storage/unit1/distributed The items deposited in this storage unit originating from another storage unit but not rejected. Can be null, if all values are 0. default: null
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. Can be null, if all values are 0. default: null
transport List of items moved to transport by this transaction or command. default: null
stacker List of items moved to stacker by this transaction or command. default: null

Event Messages

- Storage.StorageErrorEvent
- CashManagement.NoteErrorEvent
- 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 set target to *null*. In this case the Device will determine where to move any items found.

When end-to-end (E2E) security is being enforced by a device, if this command would result in notes being moved to a position where they would be accessible, this command will be blocked from executing. The exact definition of 'accessible' is hardware dependent but, for example, any position outside the safe, or any position where a attacker could access the cash should mean the command is blocked. Any attempt to execute the command will complete with the completion code *unsupportedCommand*. This is required because there is currently no E2E security defined for this command, and if the command were permitted it would be possible to extract cash and bypass E2E security.

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.

If performing a Mixed Media transaction and media items are to be moved to a storage unit or units, the requested unit(s) must support types appropriate to the media being stored.

Command Message

Payload (version 2.0)	Type	Required
{		
"position": {	object, null	
"target": "singleUnit",	string	✓
"unit": "unit4",	string, null	
"index": 1	integer, null	
}		
}		

Properties
position Defines where items are to be or have been moved to as one of the following: • A single storage unit, further specified by *unit*. • Internal areas of the device. If the *retract* area is used, the *index* property has to be provided. • An output position. This may be null: • On command data if the Service is to detrermine where items are to be moved • On completion or events if no items were moved default: null
position/target This property specifies the target where items are to be moved to. Following values are possible: • `singleUnit` - Move the items to a single storage unit defined by *unit*. • `retract` - Move the items to a retract storage unit. • `transport` - Move the items to the transport. • `stacker` - Move the items to the intermediate stacker area. • `reject` - Move the items to a reject storage unit. • `itemCassette` - Move the items to the storage units which would be used during a Cash In transaction including recycling storage units. • `cashIn` - Move the items to the storage units which would be used during a Cash In transaction but not including recycling storage units. • `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/unit If *target* is set to *singleUnit*, this property 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]+$` default: null
position/index If *target* 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. Property value constraints: `minimum: 1` default: null

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "cashUnitError",`	string, null	

Payload (version 2.0)	Type	Required
"storage": {	object, null	
"unit1": {	object, null	
"retractOperations": 15,	integer, null	
"deposited": {	object, null	
"unrecognized": 5,	integer, null	
"type20USD1": {	object, null	
"fit": 15,	integer, null	
"unfit": 0,	integer, null	
"suspect": 0,	integer, null	
"counterfeit": 0,	integer, null	
"inked": 0	integer, null	
},		
"type50USD1": See storage/unit1/deposited/type20USD1 properties	object, null	
},		
"retracted": See storage/unit1/deposited properties	object, null	
"rejected": See storage/unit1/deposited properties	object, null	
"distributed": See storage/unit1/deposited properties	object, null	
"transport": See storage/unit1/deposited properties	object, null	
},		
"unit2": See storage/unit1 properties	object, null	
},		
"transport": See storage/unit1/deposited properties	object, null	
"stacker": See storage/unit1/deposited properties	object, null	
}		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null
storage 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. default: null
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]+$` default: null
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. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
storage/unit1/deposited The items deposited in the storage unit during a Cash In transaction. Can be null, if all values are 0. default: null
storage/unit1/deposited/unrecognized Count of unrecognized items handled by the cash interface. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/deposited/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification. default: null
storage/unit1/deposited/type20USD1/fit Count of genuine cash items which are fit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null

Properties
storage/unit1/deposited/type20USD1/unfit Count of genuine cash items which are unfit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/deposited/type20USD1/suspect Count of suspected counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/deposited/type20USD1/counterfeit Count of counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/deposited/type20USD1/inked Count of cash items which have been identified as ink stained. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
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*. Can be null, if all values are 0. default: null
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. Can be null, if all values are 0. default: null
storage/unit1/distributed The items deposited in this storage unit originating from another storage unit but not rejected. Can be null, if all values are 0. default: null
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. Can be null, if all values are 0. default: null
transport List of items moved to transport by this transaction or command. default: null
stacker List of items moved to stacker by this transaction or command. default: null

Event Messages

- Storage.StorageErrorEvent
- CashManagement.InfoAvailableEvent

6.2.11 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 2.0)	Type	Required
{		
"unit": "unit1",	string	✓
"numOfBills": 40,	integer, null	
"position": {	object, null	
"target": "singleUnit",	string	✓
"unit": "unit4",	string, null	
"index": 1	integer, null	
}		
}		

Properties

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 null, 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: 1
```

default: null

position

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

- A single storage unit, further specified by *unit*.
- Internal areas of the device. If the *retract* area is used, the *index* property has to be provided.
- An output position.

This may be null:

- On command data if the Service is to detrermine where items are to be moved
- On completion or events if no items were moved

default: null

Properties
position/target This property specifies the target where items are to be moved to. Following values are possible: • `singleUnit` - Move the items to a single storage unit defined by *unit*. • `retract` - Move the items to a retract storage unit. • `transport` - Move the items to the transport. • `stacker` - Move the items to the intermediate stacker area. • `reject` - Move the items to a reject storage unit. • `itemCassette` - Move the items to the storage units which would be used during a Cash In transaction including recycling storage units. • `cashIn` - Move the items to the storage units which would be used during a Cash In transaction but not including recycling storage units. • `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/unit If *target* is set to *singleUnit*, this property 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]+$` default: null
position/index If *target* 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. Property value constraints: `minimum: 1` default: null

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "cashUnitError",`	string, null	
`"result": {`	object, null	
`"unit": "unit1",`	string	✓
`"numOfBills": 20,`	integer	
`"position": {`	object, null	
`"target": "singleUnit",`	string	✓
`"unit": "unit4",`	string, null	
`"index": 1`	integer, null	
`}`		

Payload (version 2.0)	Type	Required
} }		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

result

The result of the command, detailing where items were moved from and to. May be null if no items were moved.

default: null

result/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]+$
```

result/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
```

default: 0

result/position

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

- A single storage unit, further specified by *unit*.
- Internal areas of the device. If the *retract* area is used, the *index* property has to be provided.
- An output position.

This may be null:

- On command data if the Service is to detrermine where items are to be moved
- On completion or events if no items were moved

default: null

Properties
result/position/target This property specifies the target where items are to be moved to. Following values are possible: • `singleUnit` - Move the items to a single storage unit defined by *unit*. • `retract` - Move the items to a retract storage unit. • `transport` - Move the items to the transport. • `stacker` - Move the items to the intermediate stacker area. • `reject` - Move the items to a reject storage unit. • `itemCassette` - Move the items to the storage units which would be used during a Cash In transaction including recycling storage units. • `cashIn` - Move the items to the storage units which would be used during a Cash In transaction but not including recycling storage units. • `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.
result/position/unit If *target* is set to *singleUnit*, this property 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]+$` default: null
result/position/index If *target* 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. Property value constraints: `minimum: 1` default: null

Event Messages

- Storage.StorageErrorEvent
- CashManagement.NoteErrorEvent
- CashManagement.InfoAvailableEvent

6.3 Event Messages

6.3.1 CashManagement.NoteErrorEvent

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

Event Message

Payload (version 2.0)	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.

6.3.2 CashManagement.InfoAvailableEvent

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

Event Message

Payload (version 2.0)	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
```

6.3.3 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 2.0)	Type	Required
{		
"itemNumberList": {	object, null	
"*unit1*": {	object, null	
"retractOperations": 15,	integer, null	
"deposited": {	object, null	
"unrecognized": 5,	integer, null	
"*type20USD1*": {	object, null	
"fit": 15,	integer, null	
"unfit": 0,	integer, null	
"suspect": 0,	integer, null	
"counterfeit": 0,	integer, null	
"inked": 0	integer, null	
},		
"*type50USD1*": See *itemNumberList/unit1/deposited/type20USD1* properties	object, null	
},		
"retracted": See itemNumberList/unit1/deposited properties	object, null	
"rejected": See itemNumberList/unit1/deposited properties	object, null	
"distributed": See itemNumberList/unit1/deposited properties	object, null	
"transport": See itemNumberList/unit1/deposited properties	object, null	
},		
"*unit2*": See *itemNumberList/unit1* properties	object, null	
},		
"reason": "retractFailure"	string	✓
}		

Properties
itemNumberList 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. default: null
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]+$` default: null
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. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
itemNumberList/unit1/deposited The items deposited in the storage unit during a Cash In transaction. Can be null, if all values are 0. default: null
itemNumberList/unit1/deposited/unrecognized Count of unrecognized items handled by the cash interface. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
itemNumberList/unit1/deposited/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification. default: null
itemNumberList/unit1/deposited/type20USD1/fit Count of genuine cash items which are fit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
itemNumberList/unit1/deposited/type20USD1/unfit Count of genuine cash items which are unfit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
itemNumberList/unit1/deposited/type20USD1/suspect Count of suspected counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null

Properties
itemNumberList/unit1/deposited/type20USD1/counterfeit Count of counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
itemNumberList/unit1/deposited/type20USD1/inked Count of cash items which have been identified as ink stained. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
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*. Can be null, if all values are 0. default: null
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. Can be null, if all values are 0. default: null
itemNumberList/unit1/distributed The items deposited in this storage unit originating from another storage unit but not rejected. Can be null, if all values are 0. default: null
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. Can be null, if all values are 0. default: null
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.

6.4 Unsolicited Messages

6.4.1 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.

Unsolicited Message

Payload (version 2.0)	Type	Required
{		
"tellerID": 0	integer	✓
}		

Properties

tellerID

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

Property value constraints:

```
minimum: 0
```

6.4.2 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 2.0)	Type	Required
{		
"position": "inLeft",	string	✓
"additionalBunches": "1"	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.

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"

6.4.3 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 2.0)	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.

6.4.4 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 2.0)	Type	Required
{		
"position": "inLeft",	string	✓
"additionalBunches": "1"	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.

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"

6.4.5 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 *target* will be *null*.

Unsolicited Message

Payload (version 2.0)	Type	Required
{		
"target": "singleUnit",	string	✓
"unit": "unit4",	string, null	
"index": 1	integer, null	
}		

Properties

target

This property specifies the target where items are to be moved to. Following values are possible:

- `singleUnit` - Move the items to a single storage unit defined by *unit*.
- `retract` - Move the items to a retract storage unit.
- `transport` - Move the items to the transport.
- `stacker` - Move the items to the intermediate stacker area.
- `reject` - Move the items to a reject storage unit.
- `itemCassette` - Move the items to the storage units which would be used during a Cash In transaction including recycling storage units.
- `cashIn` - Move the items to the storage units which would be used during a Cash In transaction but not including recycling storage units.
- `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.

unit

If *target* is set to *singleUnit*, this property 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]+$
```

default: null

index

If *target* 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.

Property value constraints:

```
minimum: 1
```

default: null

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 2.0)	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 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"mixes": {`	object, null	
`"mix1": {`	object	
`"type": "algorithm",`	string	✓
`"algorithm": "minimumBills",`	string, null	
`"name": "Minimum Bills"`	string, null	
`},`		
`"mixIndividual": See mixes/mix1 properties`	object	
`}`		
`}`		

Properties

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. Can be null if empty.

default: null

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.

Properties
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. null 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$\|^[A-Za-z0-9]*$` default: null
mixes/mix1/name Name of the table or algorithm used. May be null if not defined. default: null

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 2.0)	Type	Required
{		
"mix": "mixTable21"	string	✓
}		

Properties

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

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "invalidMix",	string, null	
"mixNumber": 21,	integer, null	
"name": "House mix 21",	string, null	
"mixRows": [{	array (object)	✓
"amount": 0.30,	number	✓
"mix": [{	array (object)	✓
"value": 0.05,	number	✓
"count": 6	integer	✓
}]		
}]		
}		

Properties

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

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

default: null

mixNumber
Number identifying the house mix table (optional).
Property value constraints:

```
minimum: 1
```

default: null

name
Name of the house mix table. Null if not defined.
default: null

mixRows
Array of rows of the mix table.

Properties
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/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 2.0)	Type	Required
`{`		
`"position": "outDefault",`	string	
`"nonce": "646169ECDD0E440C2CECC8DDD7C27C22"`	string, null	
`}`		

Properties

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"

nonce

A nonce value to be used when creating the end-to-end security token in the response. If no token is requested this property should be null. See the generic end-to-end security documentation for more details.

Property value constraints:

```
pattern: ^[0-9A-F]{32}$|^[0-9]*$
```

default: null

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "unsupportedPosition",	string, null	
"denomination": {	object, null	
"currencies": {	object	✓
"EUR": 10.00,	number	
"USD": See denomination/currencies/EUR	number	
},		
"values": {	object	✓
"unit1": 5,	integer	
"unit2": See denomination/values/unit1	integer	
},		
"cashBox": {	object, null	
"currencies": See denomination/currencies properties	object	
}		
},		
"presentState": "presented",	string	✓
"token": "NONCE=1414,TOKENFORMAT=1,TOKENLENGTH ..."	string, null	
}		

Properties

errorCode

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

- `unsupportedPosition` - The specified output position is not supported.

default: null

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.

May be null where no items were dispensed.

default: null

denomination/currencies

List of currency and amount combinations for denomination requests or output. There will be one entry for each currency in the 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/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. default: null
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. Only provided if the command message contained the nonce property. 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` default: null

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. Requests are validated against the number of items and availability of each requested storage unit.

The request contains one of the following items:

- A *service mix* where the amount to be denominated is provided and the Service determines the mix of items to meet the request. The algorithm or mix table used to determine the mix is specified and may include a *partial* list of items from specific storage units which must be included in the denomination. A partial mix must be specified if items of no currency value are to be included such as coupons or documents.
- An *application mix* where the number of items from each storage unit in the denomination is pre-determined and the request confirms whether it is possible to dispense that mix of items.

Multiple currencies may be specified using *currencies*.

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

Command Message

Payload (version 2.0)	Type	Required
{		
"request": {	object	✓
"denomination": {	object	✓
"app": {	object, null	
"currencies": {	object	✓
"*EUR*": 10.00,	number	
"*USD*": See *request/denomination/app/currencies/EUR*	number	
},		
"counts": {	object	✓
"*unit1*": 5,	integer	
"*unit2*": See *request/denomination/app/counts/unit1*	integer	
},		
"cashBox": {	object, null	
"currencies": See request/denomination/app/currencies properties	object	
}		
},		
"service": {	object, null	
"currencies": See request/denomination/app/currencies properties	object	✓
"partial": See request/denomination/app/counts properties	object, null	
"mix": "mix1",	string	✓
"cashBox": See request/denomination/app/cashBox properties	object, null	

Payload (version 2.0)	Type	Required
} },		
"tellerID": 0	integer, null	
} }		

Properties

request
The request to be denominated.

request/denomination
The items to be denominated or dispensed as appropriate. The mix of items is either determined by the Service or the Application.
Property value constraints:
```
minProperties: 1
maxProperties: 1
```

request/denomination/app
Specifies a denomination request where the application determines the mix of items based on the inputs.
default: null

request/denomination/app/currencies
List of currency and amount combinations for denomination requests or output. There will be one entry for each currency in the denomination.

request/denomination/app/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
```

request/denomination/app/counts
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.

request/denomination/app/counts/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
```

request/denomination/app/cashBox
Only applies to Teller Dispensers. Amount to be paid from the teller's cash box.
default: null

request/denomination/service
Specifies a denomination request where the Service is to determine the mix of items based on the inputs. The mix may require specific items to be included using *partial*.
default: null

Properties
request/denomination/service/partial This list specifies items which must be included in a denominate or dispense request. Mixes may only be valid if they contain at least these specified items. This may be null if there are no minimum requirements. 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. default: null
request/denomination/service/mix Mix algorithm or house mix table to be used as defined by mixes reported by CashDispenser.GetMixTypes. Property value constraints: `pattern: ^mix[0-9A-Za-z]+$`
request/tellerID Only applies to Teller Dispensers, null if not applicable. Identification of teller. Property value constraints: `minimum: 0` default: null

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "invalidCurrency",	string, null	
"result": {	object, null	
"currencies": {	object	✓
"*EUR*": 10.00,	number	
"*USD*": See *result/currencies/EUR*	number	
},		
"values": {	object	✓
"*unit1*": 5,	integer	
"*unit2*": See *result/values/unit1*	integer	
},		
"cashBox": {	object, null	
"currencies": See result/currencies properties	object	
}		
}		
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

result

Specifies the denomination if successful. May be null where a denomination could not be determined.

default: null

result/currencies

List of currency and amount combinations for denomination requests or output. There will be one entry for each currency in the denomination.

result/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
```

result/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.

result/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
```

result/cashBox

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

default: null

Event Messages

- Storage.StorageErrorEvent

7.2.5 CashDispenser.Dispense

This command dispenses items from the storage units. See CashDispenser.Denominate for a description of how the denomination may be specified. The items are moved to the intermediate stacker if the device has one, and a CashDispenser.Present command is used to present the items to the user. If the device does not have an intermediate stacker the items will be presented to the user using this command. The *position* property in the command data specifies which output position the items are intended to be presented to, and applies whether or not the items are actually presented by this command as items may need to be stacked in a particular position ready for presentation at the intended output position.

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, *position* can be set to *outDefault*. If this is the case the *tellerID* is used to perform the dispense operation to the assigned teller position.

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* property 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" 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 2.0)	Type	Required
{		
"denomination": {	object	✓
"denomination": {	object	✓
"app": {	object, null	
"currencies": {	object	✓
"*EUR*": 10.00,	number	
"*USD*": See *denomination/denomination/app/currencies/EUR*	number	
},		

Payload (version 2.0)	Type	Required
"counts": {	object	✓
"unit1": 5,	integer	
"unit2": See denomination/denomination/app/counts/unit1	integer	
},		
"cashBox": {	object, null	
"currencies": See denomination/denomination/app/currencies properties	object	
}		
},		
"service": {	object, null	
"currencies": See denomination/denomination/app/currencies properties	object	✓
"partial": See denomination/denomination/app/counts properties	object, null	
"mix": "mix1",	string	✓
"cashBox": See denomination/denomination/app/cashBox properties	object, null	
}		
},		
"tellerID": 0	integer, null	
},		
"position": "outDefault",	string	
"token": "NONCE=254611E63B2531576314E86527338D61, ..."	string, null	
}		

Properties

denomination

Denomination object describing the contents of the dispense operation.

denomination/denomination

The items to be denominated or dispensed as appropriate. The mix of items is either determined by the Service or the Application.

Property value constraints:

```
minProperties: 1
maxProperties: 1
```

denomination/denomination/app

Specifies a denomination request where the application determines the mix of items based on the inputs.

default: null

denomination/denomination/app/currencies

List of currency and amount combinations for denomination requests or output. There will be one entry for each currency in the denomination.

Properties

denomination/denomination/app/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/denomination/app/counts 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/app/counts/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/app/cashBox Only applies to Teller Dispensers. Amount to be paid from the teller's cash box. default: null
denomination/denomination/service Specifies a denomination request where the Service is to determine the mix of items based on the inputs. The mix may require specific items to be included using *partial*. default: null
denomination/denomination/service/partial This list specifies items which must be included in a denominate or dispense request. Mixes may only be valid if they contain at least these specified items. This may be null if there are no minimum requirements. 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. default: null
denomination/denomination/service/mix Mix algorithm or house mix table to be used as defined by mixes reported by CashDispenser.GetMixTypes. Property value constraints: `pattern: ^mix[0-9A-Za-z]+$`
denomination/tellerID Only applies to Teller Dispensers, null if not applicable. Identification of teller. Property value constraints: `minimum: 0` default: null

Properties
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"
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* property. 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` default: null

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "invalidCurrency",`	string, null	

Payload (version 2.0)	Type	Required
"denomination": {	object, null	
"currencies": {	object	✓
"EUR": 10.00,	number	
"USD": See denomination/currencies/EUR	number	
},		
"values": {	object	✓
"unit1": 5,	integer	
"unit2": See denomination/values/unit1	integer	
},		
"cashBox": {	object, null	
"currencies": See denomination/currencies properties	object	
}		
},		
"bunches": "1",	string	
"storage": {	object, null	
"in": {	object, null	
"unit1": {	object, null	
"retractOperations": 15,	integer, null	
"deposited": {	object, null	
"unrecognized": 5,	integer, null	
"type20USD1": {	object, null	
"fit": 15,	integer, null	
"unfit": 0,	integer, null	
"suspect": 0,	integer, null	
"counterfeit": 0,	integer, null	
"inked": 0	integer, null	
},		
"type50USD1": See storage/in/unit1/deposited/type20USD1 properties	object, null	
},		

Payload (version 2.0)	Type	Required
"retracted": See storage/in/unit1/deposited properties	object, null	
"rejected": See storage/in/unit1/deposited properties	object, null	
"distributed": See storage/in/unit1/deposited properties	object, null	
"transport": See storage/in/unit1/deposited properties	object, null	
},		
"*unit2*": See *storage/in/unit1* properties	object, null	
},		
"out": {	object	✓
"*unit3*": {	object, null	
"presented": See storage/in/unit1/deposited properties	object, null	
"rejected": See storage/in/unit1/deposited properties	object, null	
"distributed": See storage/in/unit1/deposited properties	object, null	
"unknown": See storage/in/unit1/deposited properties	object, null	
"stacked": See storage/in/unit1/deposited properties	object, null	
"diverted": See storage/in/unit1/deposited properties	object, null	
"transport": See storage/in/unit1/deposited properties	object, null	
},		
"*unit4*": See *storage/out/unit3* properties	object, null	
}		
}		
}		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. • `safeDoorOpen` - The safe door is open. This device requires the safe door to be closed in order to perform this operation. (See Common.Status) property. default: null
denomination Denomination object describing the contents of the dispense operation. default: null
denomination/currencies List of currency and amount combinations for denomination requests or output. There will be one entry for each currency in the 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`

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.

Properties

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. default: null
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. This property is null if no items are moved. default: null
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. default: null
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]+$` default: null
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. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
storage/in/unit1/deposited The items deposited in the storage unit during a Cash In transaction. Can be null, if all values are 0. default: null
storage/in/unit1/deposited/unrecognized Count of unrecognized items handled by the cash interface. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null

Properties
storage/in/unit1/deposited/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification. default: null
storage/in/unit1/deposited/type20USD1/fit Count of genuine cash items which are fit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/in/unit1/deposited/type20USD1/unfit Count of genuine cash items which are unfit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/in/unit1/deposited/type20USD1/suspect Count of suspected counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/in/unit1/deposited/type20USD1/counterfeit Count of counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/in/unit1/deposited/type20USD1/inked Count of cash items which have been identified as ink stained. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
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*. Can be null, if all values are 0. default: null
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. Can be null, if all values are 0. default: null
storage/in/unit1/distributed The items deposited in this storage unit originating from another storage unit but not rejected. Can be null, if all values are 0. default: null

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. Can be null, if all values are 0. default: null
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]+$` default: null
storage/out/unit3/presented The items dispensed from this storage unit which are or were customer accessible. Will be null if no items were presented. default: null
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. Will be null if no items were rejected. default: null
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. Will be null if no items were distributed. default: null
storage/out/unit3/unknown The items dispensed from this storage unit which moved to an unknown position. Will be null if no items were unknown. default: null
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. Will be null if no items were stacked. default: null
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. Will be null if no items were diverted. default: null
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. Will be null if no items apply. default: null

Event Messages

- CashDispenser.DelayedDispenseEvent
- CashDispenser.StartDispenseEvent

- Storage.StorageErrorEvent
- 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.

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 2.0)	Type	Required
{		
"position": "outDefault"	string	
}		

Properties

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 2.0)	Type	Required
{		
"errorCode": "shutterNotOpen",	string, null	
"position": {	object, null	
"position": "inLeft",	string	✓
"additionalBunches": "1"	string	
}		
}		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null
position Provides information about the presented items. May be null if no items were presented. default: null
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"

Event Messages

- CashManagement.InfoAvailableEvent

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 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "cashUnitError"`	string, null	
`}`		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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).

default: null

Event Messages

- 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 2.0)	Type	Required
`{`		
`"mixNumber": 21,`	integer, null	
`"name": "House mix 21",`	string, null	
`"mixRows": [{`	array (object)	✓
`"amount": 0.30,`	number	✓
`"mix": [{`	array (object)	✓
`"value": 0.05,`	number	✓
`"count": 6`	integer	✓
`}]`		
`}]`		
`}`		

Properties

mixNumber
Number identifying the house mix table (optional).
Property value constraints:
`minimum: 1`
default: null

name
Name of the house mix table. Null if not defined.
default: null

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/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 2.0)	Type	Required
`{`		
`"errorCode": "invalidMixNumber"`	string, null	
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null

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.

When end-to-end (E2E) security is being enforced by a device, if this command would result in notes being moved to a position where they would be accessible, this command will be blocked from executing. The exact definition of 'accessible' is hardware dependent but, for example, any position outside the safe, or any position where a attacker could access the cash should mean the command is blocked. Any attempt to execute the command will complete with the completion code *unsupportedCommand*. This is required because there is currently no E2E security defined for this command, and if the command were permitted it would be possible to extract cash and bypass E2E security.

Command Message

Payload (version 2.0)	Type	Required
{		
"target": {	object, null	
"target": "singleUnit",	string	✓
"unit": "unit4",	string, null	
"index": 1	integer, null	
}		
}		

Properties

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

- A single storage unit, further specified by *unit*.
- Internal areas of the device. If the *retract* area is used, the *index* property has to be provided.
- An output position.

This may be null:

- On command data if the Service is to detrermine where items are to be moved
- On completion or events if no items were moved

default: null

Properties
target/target This property specifies the target where items are to be moved to. Following values are possible: • `singleUnit` - Move the items to a single storage unit defined by *unit*. • `retract` - Move the items to a retract storage unit. • `transport` - Move the items to the transport. • `stacker` - Move the items to the intermediate stacker area. • `reject` - Move the items to a reject storage unit. • `itemCassette` - Move the items to the storage units which would be used during a Cash In transaction including recycling storage units. • `cashIn` - Move the items to the storage units which would be used during a Cash In transaction but not including recycling storage units. • `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.
target/unit If *target* is set to *singleUnit*, this property 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]+$` default: null
target/index If *target* 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. Property value constraints: `minimum: 1` default: null

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "cashUnitError"`	string, null	
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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.StorageErrorEvent events 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. default: null

Event Messages

- Storage.StorageErrorEvent
- 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.

When end-to-end (E2E) security is being enforced by a device this command will be blocked from executing. Any attempt to execute the command will complete with the completion code *unsupportedData*. This is required because there is currently no E2E security defined for this command, and if the command were permitted it would be possible to extract cash and bypass E2E security.

Command Message

Payload (version 2.0)	Type	Required
{		
"unit": "unit1",	string, null	
"position": "outDefault"	string	
}		

Properties

unit

Specifies the unit to empty. If this property is null, all units are emptied. Following values are possible:

- `<storage unit identifier>` - The storage unit to be emptied as identifier.

Property value constraints:

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

default: null

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 2.0)	Type	Required
{		
"errorCode": "cashUnitError",	string, null	
"countedCashUnits": {	object, null	
"*unit1*": {	object	
"dispensed": 100,	integer	✓
"counted": 100,	integer	✓

Payload (version 2.0)	Type	Required
"replenishmentStatus": "ok",	string, null	
"status": "ok"	string, null	
},		
"*unit2*": See *countedCashUnits/unit1* properties	object	
}		
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

(See Common.Status) property.

- `exchangeActive` - The device is in an exchange state (see Storage.StartExchange).

default: null

countedCashUnits

List of counted storage unit objects. Will be null if no units were counted.

default: null

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
```

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. May be null in events if not changing, otherwise 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.

default: null

Properties
countedCashUnits/unit1/status The state of the unit. This property may be null in events if the state did not change, otherwise 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. default: null

Event Messages

- Storage.StorageErrorEvent
- 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 2.0)	Type	Required
{		
"action": "start"	string	✓
}		

Properties

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, 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 2.0)	Type	Required
{		
"errorCode": "exchangeActive"	string, null	
}		

Properties

errorCode

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

- `exchangeActive` - The device is in an exchange state (see Storage.StartExchange).

default: null

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 2.0)	Type	Required
{		
"delay": 0.10	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 2.0)
`This message does not define any properties.`

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 2.0)	Type	Required
{		
"currencies": {	object	✓
"EUR": 10.00,	number	
"USD": See currencies/EUR	number	
},		
"values": {	object	✓
"unit1": 5,	integer	
"unit2": See values/unit1	integer	
},		
"cashBox": {	object, null	
"currencies": See currencies properties	object	
}		
}		

Properties

currencies

List of currency and amount combinations for denomination requests or output. There will be one entry for each currency in the 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
```

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. default: null

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 2.0)
This message does not define any properties.

Completion Message

Payload (version 2.0)	Type	Required
{		
"status": "unknown",	string	
"numOfRefused": 0,	integer, null	
"noteNumberList": {	object, null	
"unrecognized": 5,	integer, null	
"*type20USD1*": {	object, null	
"fit": 15,	integer, null	
"unfit": 0,	integer, null	
"suspect": 0,	integer, null	
"counterfeit": 0,	integer, null	
"inked": 0	integer, null	
},		
"*type50USD1*": See *noteNumberList/type20USD1* properties	object, null	
}		
}		

Properties

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.

default: "unknown"

Properties
numOfRefused Specifies the number of items refused during the currently active or most recently ended cash-in transaction period. May be null if no items were refused. Property value constraints: `minimum: 0` default: null
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. It will be null if no banknotes were accepted. Includes any identified notes. default: null
noteNumberList/unrecognized Count of unrecognized items handled by the cash interface. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
noteNumberList/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification. default: null
noteNumberList/type20USD1/fit Count of genuine cash items which are fit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
noteNumberList/type20USD1/unfit Count of genuine cash items which are unfit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
noteNumberList/type20USD1/suspect Count of suspected counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
noteNumberList/type20USD1/counterfeit Count of counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null

Properties
noteNumberList/type20USD1/inked Count of cash items which have been identified as ink stained. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null

Event Messages

None

8.1.2 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 2.0)	Type	Required
{		
"source": "unit2"	string	✓
}		

Properties

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

Completion Message

Payload (version 2.0)	Type	Required
{		
"targets": [{	array (object), null	
"target": "unit1"	string	✓
}]		
}		

Properties

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

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

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

Event Messages

None

8.1.3 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 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"deviceLockStatus": "lockUnknown",`	string	
`"unitLock": [{`	array (object), null	
`"storageUnit": "unit1",`	string	✓
`"unitLockStatus": "lockUnknown"`	string	
`}]`		
`}`		

Properties

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.

default: "lockUnknown"

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.

default: null

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.

default: "lockUnknown"

Event Messages

None

8.1.4 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 2.0)	Type	Required
{		
"cashUnitTarget": "unit2"	string	✓
}		

Properties

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

Completion Message

Payload (version 2.0)	Type	Required
{		
"depleteSources": [{	array (object), null	
"cashUnitSource": "unit1"	string	✓
}]		
}		

Properties

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

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

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

Event Messages

None

8.1.5 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 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"position": "outDefault",`	string	
`"presentState": "unknown",`	string	
`"additionalBunches": "unknown",`	string	
`"bunchesRemaining": 0,`	integer, null	
`"returnedItems": {`	object, null	
`"unrecognized": 5,`	integer, null	
`"type20USD1": {`	object, null	
`"fit": 15,`	integer, null	
`"unfit": 0,`	integer, null	
`"suspect": 0,`	integer, null	
`"counterfeit": 0,`	integer, null	
`"inked": 0`	integer, null	
`},`		
`"type50USD1": See returnedItems/type20USD1 properties`	object, null	
`},`		
`"totalReturnedItems": See returnedItems properties`	object, null	
`"remainingItems": See returnedItems properties`	object, null	
`}`		

Properties

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
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. default: "unknown"
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. default: "unknown"
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 null 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`. Property value constraints: `minimum: 0` default: null
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. default: null
returnedItems/unrecognized Count of unrecognized items handled by the cash interface. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
returnedItems/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification. default: null
returnedItems/type20USD1/fit Count of genuine cash items which are fit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
returnedItems/type20USD1/unfit Count of genuine cash items which are unfit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null

Properties
returnedItems/type20USD1/suspect Count of suspected counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
returnedItems/type20USD1/counterfeit Count of counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
returnedItems/type20USD1/inked Count of cash items which have been identified as ink stained. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
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. default: null
remainingItems Array of counts of banknotes on the intermediate stacker or transport which have not been yet moved to the output position. default: null

Event Messages

None

8.1.6 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 2.0)	**Type**	**Required**
{		
"tellerID": 0,	integer, null	
"useRecycleUnits": true,	boolean	
"outputPosition": "outDefault",	string	
"inputPosition": "inDefault",	string	
"totalItemsLimit": 0,	integer	
"amountLimit": [{	array (object), null	
"currency": "USD",	string	✓
"value": 20.00	number, null	
}]		
}		

Properties

tellerID

Identification of teller. This property is not applicable to Self-Service devices and can therefore be null.

Property value constraints:

```
minimum: 0
```

default: null

Properties
useRecycleUnits Specifies whether or not the recycle storage units should be used when items are cashed in on a successful CashAcceptor.CashInEnd command. This property 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. If a value of null is specified for a currency, there is no amount limit applied to the currency. default: null
amountLimit/currency ISO 4217 currency identifier [Ref. cashmanagement-1]. Property value constraints: `pattern: ^[A-Z]{3}$`

Properties
amountLimit/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. May be null in command data or events if not being modified. Property value constraints: `minimum: 0` default: null

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "invalidTellerId"`	string, null	
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. (See Common.Status) property. default: null

Event Messages

None

8.1.7 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 property 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 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 this command opens the shutter at the start of the command and closes it once bills are inserted.

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.

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 2.0)
This message does not define any properties.

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "cashUnitError",	string, null	
"items": {	object, null	
"unrecognized": 5,	integer, null	
"*type20USD1*": {	object, null	
"fit": 15,	integer, null	
"unfit": 0,	integer, null	
"suspect": 0,	integer, null	
"counterfeit": 0,	integer, null	
"inked": 0	integer, null	
},		
"*type50USD1*": See *items/type20USD1* properties	object, null	
}		
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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. (See Common.Status) property.
- `foreignItemsDetected` - Foreign items have been detected inside the input position.
- `shutterNotOpen` - Shutter failed to open.

default: null

items

Items detected during the command. May be null if no items were detected. This information is not cumulative over multiple *CashIn* commands.

default: null

Properties
items/unrecognized Count of unrecognized items handled by the cash interface. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
items/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification. default: null
items/type20USD1/fit Count of genuine cash items which are fit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
items/type20USD1/unfit Count of genuine cash items which are unfit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
items/type20USD1/suspect Count of suspected counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
items/type20USD1/counterfeit Count of counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
items/type20USD1/inked Count of cash items which have been identified as ink stained. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null

Event Messages

- Storage.StorageErrorEvent
- CashAcceptor.InputRefuseEvent
- CashManagement.NoteErrorEvent
- CashAcceptor.SubCashInEvent
- CashManagement.InfoAvailableEvent
- CashAcceptor.InsertItemsEvent

8.1.8 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 completion payload. If no items are available for cash-in for any other reason, the *noItems* error code is returned.

Command Message

Payload (version 2.0)
This message does not define any properties.

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "cashUnitError",	string, null	
"storage": {	object, null	
"*unit1*": {	object, null	
"retractOperations": 15,	integer, null	
"deposited": {	object, null	
"unrecognized": 5,	integer, null	
"*type20USD1*": {	object, null	
"fit": 15,	integer, null	
"unfit": 0,	integer, null	
"suspect": 0,	integer, null	
"counterfeit": 0,	integer, null	
"inked": 0	integer, null	
},		
"*type50USD1*": See *storage/unit1/deposited/type20USD1* properties	object, null	
},		
"retracted": See storage/unit1/deposited properties	object, null	

Payload (version 2.0)	Type	Required
"rejected": See storage/unit1/deposited properties	object, null	
"distributed": See storage/unit1/deposited properties	object, null	
"transport": See storage/unit1/deposited properties	object, null	
},		
"*unit2*": See *storage/unit1* properties	object, null	
}		
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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. (See Common.Status) property.

default: null

storage

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.

default: null

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]+$
```

default: null

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. May be null in command data and events if not changing.

Property value constraints:

```
minimum: 0
```

default: null

storage/unit1/deposited

The items deposited in the storage unit during a Cash In transaction. Can be null, if all values are 0.

default: null

Properties
storage/unit1/deposited/unrecognized Count of unrecognized items handled by the cash interface. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/deposited/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification. default: null
storage/unit1/deposited/type20USD1/fit Count of genuine cash items which are fit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/deposited/type20USD1/unfit Count of genuine cash items which are unfit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/deposited/type20USD1/suspect Count of suspected counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/deposited/type20USD1/counterfeit Count of counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/deposited/type20USD1/inked Count of cash items which have been identified as ink stained. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
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*. Can be null, if all values are 0. default: null
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. Can be null, if all values are 0. default: null

Properties
storage/unit1/distributed The items deposited in this storage unit originating from another storage unit but not rejected. Can be null, if all values are 0. default: null
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. Can be null, if all values are 0. default: null

Event Messages

- Storage.StorageErrorEvent
- CashManagement.InfoAvailableEvent
- CashManagement.NoteErrorEvent

8.1.9 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 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 opens the shutter and it is closed when all items are 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.

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 completion payload. 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 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "cashUnitError",`	string, null	
`"storage": {`	object, null	
`"unit1": {`	object, null	
`"retractOperations": 15,`	integer, null	
`"deposited": {`	object, null	
`"unrecognized": 5,`	integer, null	
`"type20USD1": {`	object, null	
`"fit": 15,`	integer, null	
`"unfit": 0,`	integer, null	

Payload (version 2.0)	Type	Required
"suspect": 0,	integer, null	
"counterfeit": 0,	integer, null	
"inked": 0	integer, null	
},		
"*type50USD1*": See *storage/unit1/deposited/type20USD1* properties	object, null	
},		
"retracted": See storage/unit1/deposited properties	object, null	
"rejected": See storage/unit1/deposited properties	object, null	
"distributed": See storage/unit1/deposited properties	object, null	
"transport": See storage/unit1/deposited properties	object, null	
},		
"*unit2*": See *storage/unit1* properties	object, null	
}		
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

storage

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.

default: null

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]+$
```

default: null

Properties
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. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
storage/unit1/deposited The items deposited in the storage unit during a Cash In transaction. Can be null, if all values are 0. default: null
storage/unit1/deposited/unrecognized Count of unrecognized items handled by the cash interface. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/deposited/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification. default: null
storage/unit1/deposited/type20USD1/fit Count of genuine cash items which are fit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/deposited/type20USD1/unfit Count of genuine cash items which are unfit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/deposited/type20USD1/suspect Count of suspected counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/deposited/type20USD1/counterfeit Count of counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/deposited/type20USD1/inked Count of cash items which have been identified as ink stained. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null

Properties
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*. Can be null, if all values are 0. default: null
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. Can be null, if all values are 0. default: null
storage/unit1/distributed The items deposited in this storage unit originating from another storage unit but not rejected. Can be null, if all values are 0. default: null
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. Can be null, if all values are 0. default: null

Event Messages

- Storage.StorageErrorEvent
- CashManagement.InfoAvailableEvent

8.1.10 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 2.0)	Type	Required
{		
"items": [{	array (object)	✓
"item": "type20USD1",	string	✓
"enabled": false	boolean	✓
}]		
}		

Properties

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.
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.

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "exchangeActive"	string, null	
}		

Properties

errorCode
Specifies the error code if applicable, otherwise null. 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.
default: null

Event Messages

None

8.1.11 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 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, 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 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.

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 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "tooManyItems",	string, null	
"noteType": "type20USD1",	string, null	
"orientation": "frontTop",	string, null	
"signature": "MAA5ADgANwA2ADUANAAz ..."	string, null	
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

Properties
noteType A cash item as reported by CashManagement.GetBankNoteTypes. This is null if the item was not identified as a cash item. Property value constraints: `pattern: ^type[0-9A-Z]+$` default: null
orientation Specifies the note orientation. This property is null if the hardware is not capable to determine the 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. default: null

signature
Base64 encoded vendor specific signature data. If no signature is available or has not been requested then this is null.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`
default: null

Event Messages

- CashAcceptor.InputRefuseEvent
- CashManagement.NoteErrorEvent
- CashAcceptor.InsertItemsEvent
- CashManagement.InfoAvailableEvent

8.1.12 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 2.0)	Type	Required
{		
"loadAlways": false	boolean	
}		

Properties

loadAlways
If set to true, the Service loads the currency description data into the note reader, even if it is already loaded.
default: false

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "exchangeActive",	string, null	
"rebootNecessary": false	boolean	
}		

Properties

errorCode
Specifies the error code if applicable, otherwise null. 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.
default: null

rebootNecessary
If set to true, the machine needs a reboot before the note reader can be accessed again.
default: false

Event Messages

None

8.1.13 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 2.0)	Type	Required
{		
"referenceSignatures": [{	array (object)	✓
"noteType": "type20USD1",	string, null	
"orientation": "frontTop",	string, null	
"signature": "MAA5ADgANwA2ADUANAAz ..."	string, null	
}],		
"signatures": See referenceSignatures properties	array (object)	✓
}		

Properties

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.

referenceSignatures/noteType

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

Property value constraints:

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

default: null

Properties
referenceSignatures/orientation Specifies the note orientation. This property is null if the hardware is not capable to determine the 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. default: null

referenceSignatures/signature
Base64 encoded vendor specific signature data. If no signature is available or has not been requested then this is null.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`
default: null

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.

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "cashInActive",`	string, null	
`"signaturesIndex": [{`	array (object), null	
`"index": 0,`	integer	✓
`"confidenceLevel": 95,`	integer	
`"comparisonData": "Example comparison data."`	string, null	
`}]`		
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null
signaturesIndex Array of compare results. This is null 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 property *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. default: null
signaturesIndex/index Specifies the index (0 to *#signatures* - 1) of the matching signature from the input property *signatures*. Property value constraints: `minimum: 0`

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`
default: 0

signaturesIndex/comparisonData Vendor dependent comparison result data. This data may be used as justification for the signature match or confidence level. This property is null if no additional comparison data is returned. default: null

Event Messages

None

8.1.14 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 2.0)	Type	Required
{		
"source": "unit1",	string	✓
"replenishTargets": [{	array (object)	✓
"target": "unit1",	string	✓
"numberOfItemsToMove": 100	integer	
}]		
}		

Properties

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]+$`

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.GetStorage command) 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`
default: 0

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "cashUnitError",	string, null	

Payload (version 2.0)	Type	Required
"numberOfItemsRemoved": 20,	integer, null	
"numberOfItemsRejected": 2,	integer, null	
"replenishTargetResults": [{	array (object), null	
"target": "unit1",	string	✓
"cashItem": "type20USD1",	string, null	
"numberOfItemsReceived": 20	integer	✓
}]		
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

numberOfItemsRemoved

Total number of items removed from the source storage unit including rejected items during execution of this command. This property is null if no items were removed.

Property value constraints:

```
minimum: 1
```

default: null

numberOfItemsRejected

Total number of items rejected during execution of this command. This property is null if no items were rejected.

Property value constraints:

```
minimum: 1
```

default: null

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.

default: null

replenishTargetResults/target

Name of the storage unit (as stated by the Storage.GetStorage command) to which items have been moved.

Property value constraints:

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

Properties
replenishTargetResults/cashItem A cash item as reported by CashManagement.GetBankNoteTypes. This is null if the item was not identified as a cash item. Property value constraints: `pattern: ^type[0-9A-Z]+$` default: null
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
- CashManagement.NoteErrorEvent
- CashManagement.InfoAvailableEvent
- CashAcceptor.IncompleteReplenishEvent

8.1.15 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 2.0)	Type	Required
{		
"units": ["unit1", "unit2"]	array (string)	✓
}		

Properties

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 2.0)	Type	Required
{		
"errorCode": "invalidCashUnit"	string, null	
}		

Properties

errorCode
Specifies the error code if applicable, otherwise null. 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.

default: null

Event Messages

- Storage.StorageErrorEvent
- CashManagement.NoteErrorEvent
- CashManagement.InfoAvailableEvent

8.1.16 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 2.0)	Type	Required
{		
"deviceAction": "noLockAction",	string	
"cashUnitAction": "noLockAction",	string	
"unitLockControl": [{	array (object), null	
"storageUnit": "unit1",	string	✓
"unitAction": "lock"	string	✓
}]		
}		

Properties
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. default: "noLockAction"
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* property. • `noLockAction` - No lock/unlock action will be performed on storage units. default: "noLockAction"
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. default: null
unitLockControl/storageUnit Name of the storage unit (as stated by the Storage.GetStorage command) 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* property. 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 2.0)	Type	Required
`{`		
`"errorCode": "invalidCashUnit"`	string, null	
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null

Event Messages

- Storage.StorageErrorEvent

8.1.17 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 is *rollback* or *refuse* and where presentControl 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 2.0)	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.

default: "outDefault"

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "unsupportedPosition"`	string, null	
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null

Event Messages

None

8.1.18 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 2.0)	Type	Required
{		
"depleteSources": [{	array (object)	✓
"source": "unit1",	string	✓
"numberOfItemsToMove": 100	integer	
}],		
"cashUnitTarget": "unit1"	string	✓
}		

Properties

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
```

default: 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 2.0)	Type	Required
{		
"errorCode": "cashUnitError",	string, null	
"numberOfItemsReceived": 100,	integer	
"numberOfItemsRejected": 10,	integer	
"depleteSourceResults": [{	array (object), null	

Payload (version 2.0)	Type	Required
"cashUnitSource": "unit1",	string	✓
"cashItem": "type20USD1",	string, null	
"numberOfItemsRemoved": 0	integer	
}]		
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.IncompleteDepleteEvent will 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.

default: null

numberOfItemsReceived

Total number of items received in the target storage unit during execution of this command.

Property value constraints:

```
minimum: 0
```

default: 0

numberOfItemsRejected

Total number of items rejected during execution of this command.

Property value constraints:

```
minimum: 0
```

default: 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.

default: null

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]+$
```

depleteSourceResults/cashItem

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

Property value constraints:

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

default: null

Properties
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` default: 0

Event Messages

- Storage.StorageErrorEvent
- CashManagement.NoteErrorEvent
- CashManagement.InfoAvailableEvent
- CashAcceptor.IncompleteDepleteEvent

8.1.19 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 2.0)	Type	Required
{		
"position": "outDefault"	string	
}		

Properties

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 2.0)	Type	Required
{		
"errorCode": "unsupportedPosition"	string, null	
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

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 2.0)	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 2.0)	Type	Required
{		
"unrecognized": 5,	integer, null	
"*type20USD1*": {	object, null	
"fit": 15,	integer, null	
"unfit": 0,	integer, null	
"suspect": 0,	integer, null	
"counterfeit": 0,	integer, null	
"inked": 0	integer, null	
},		
"*type50USD1*": See *type20USD1* properties	object, null	
}		

Properties

unrecognized

Count of unrecognized items handled by the cash interface. May be null in command data and events if not changed or not to be changed.

Property value constraints:

```
minimum: 0
```

default: null

type20USD1 (example name)

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

default: null

type20USD1/fit

Count of genuine cash items which are fit for recycling. May be null in command data and events if not changed or not to be changed.

Property value constraints:

```
minimum: 0
```

default: null

type20USD1/unfit

Count of genuine cash items which are unfit for recycling. May be null in command data and events if not changed or not to be changed.

Property value constraints:

```
minimum: 0
```

default: null

type20USD1/suspect

Count of suspected counterfeit cash items. May be null in command data and events if not changed or not to be changed.

Property value constraints:

```
minimum: 0
```

default: null

Properties
type20USD1/counterfeit Count of counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
type20USD1/inked Count of cash items which have been identified as ink stained. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null

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 2.0)
This message does not define any properties.

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 2.0)	Type	Required
{		
"replenish": {	object	✓
"numberOfItemsRemoved": 20,	integer, null	
"numberOfItemsRejected": 2,	integer, null	
"replenishTargetResults": [{	array (object), null	
"target": "unit1",	string	✓
"cashItem": "type20USD1",	string, null	
"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. This property is null if no items were removed.
Property value constraints:
```
minimum: 1
```
default: null

replenish/numberOfItemsRejected
Total number of items rejected during execution of this command. This property is null if no items were rejected.
Property value constraints:
```
minimum: 1
```
default: null

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.

default: null

replenish/replenishTargetResults/target
Name of the storage unit (as stated by the Storage.GetStorage command) 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 null if the item was not identified as a cash item. Property value constraints: `pattern: ^type[0-9A-Z]+$` default: null
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 2.0)	Type	Required
{		
"deplete": {	object	✓
"numberOfItemsReceived": 100,	integer	
"numberOfItemsRejected": 10,	integer	
"depleteSourceResults": [{	array (object), null	
"cashUnitSource": "unit1",	string	✓
"cashItem": "type20USD1",	string, null	
"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
```

default: 0

deplete/numberOfItemsRejected

Total number of items rejected during execution of this command.

Property value constraints:

```
minimum: 0
```

default: 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.

default: null

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]+$
```

Properties
deplete/depleteSourceResults/cashItem A cash item as reported by CashManagement.GetBankNoteTypes. This is null if the item was not identified as a cash item. Property value constraints: `pattern: ^type[0-9A-Z]+$` default: null
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` default: 0

9. Check Interface

Check Processing Modules accept one or more media items (Checks, Giros, etc) and process these items according to application requirements. The Check Interface supports devices that can handle a single item as well as those devices that can handle bunches of items. The following are the three principal device types:

- Single Item: can accept and process a single item at a time.
- Multi-Item Feed with no stacker (known as an escrow in some environments): can accept a bunch of media from the customer but each item has to be processed fully (i.e. deposited in a storage unit or returned) before the next item can be processed.
- Multi-Item Feed with a stacker: can accept a bunch of media from the customer and all items can be processed together.

In the U.S., checks are always encoded in magnetic ink for reading by Magnetic Ink Character Recognition (MICR), and a single font is always used. In Europe some countries use MICR and some use Optical Character Recognition (OCR) character sets, with different fonts, for their checks.

The Check specification provides applications with an interface to control the following functions (depending on the capabilities of the specific underlying device):

- Capture an image of the front of an item in multiple formats and bit depths.
- Capture an image of the back of an item in multiple formats and bit depths.
- Read the code line of an item using a MICR reader.
- Read the code line of an item using OCR.
- Endorse (print text) on an item.
- Stamp an item.
- Return an item to the customer.
- Deposit an item in a storage unit.
- Retract items left by the customer.

The Check specification uses the concept of a Media-In transaction to track and control a customer's interaction with the device. A Media-In transaction consists of one or more Check.MediaIn commands. The transaction is initiated by the first *Check.MediaIn* command and remains active until the transaction is either confirmed through Check.MediaInEnd, or terminated by Check.MediaInRollback, Check.RetractMedia or Check.Reset. While a transaction is active the Check.GetTransactionStatus command reports the status of the current transaction. When a transaction is not active the *Check.GetTransactionStatus* command reports the status of the last transaction as well as some current status values.

In this the specification the terms "long edge" and "short edge" are used to describe the orientation of a check and length of its edges.

This interface is to be used together with the Storage interface to handle management of storage units.

9.1 General Information

9.1.1 References

ID	Description
check-1	OCR-A font - ANSI X3.17-1981 figure E1
check-2	OCR-B font - ANSI X3.49-1975 figure C2
check-3	E-13B MICR font - ISO 1004-1:2013
check-4	CMC7 MICR font - ISO 1004-2:2013
check-5	https://www.unicode.org/charts/PDF/U2440.pdf

9.1.2 Code Line Characters

This section describes how code line data is returned in the Check specification depending on how the code line was read:

- OCR-A font will conform to [Ref. check-1].

- OCR-B font will conform to [Ref. check-2].
- E-13B MICR font will conform to [Ref. check-3]. Note that the special E-13B banking symbols are defined by Unicode (see [Ref. check-5]), therefore E-13B code lines are provided without mapping - see Table 1 below for more details.
- CMC7 MICR font will conform to [Ref. check-4]. The special banking symbols in this font are not defined in Unicode, therefore they are mapped to standard characters as shown in Table 2 below.

In all cases unrecognized characters are reported as the REJECT/SUB character, 0x1A.

Table 1 - E-13B Special Banking Symbols

Symbol	MICR Definition	Unicode	Unicode Definition
⑆	Transit	U+2446	OCR Bank Branch Identification
⑇	Amount	U+2447	OCR Amount Of Check
⑈	On Us	U+2448	OCR Dash
⑉	Dash	U+2449	OCR Customer Account Number

Table 2 - CMC7 Special Banking Symbols

Symbol	Meaning	Mapping
	S1 - Start of Bank Account	a
	S2 - Start of the Amount field	b
	S3 - Terminate Routing	c
	S4 - Unused	d
	S5 - Transit / Routing	e

Example

Check code line	*codeline* reported in XFS4IoT
 ABCD 1234	aABCDb1234c

9.2 Command Messages

9.2.1 Check.GetTransactionStatus

This command is used to request the status of the current or last media-in transaction as well as current status values outside a transaction. A media-in transaction consists of one or more Check.MediaIn commands. A media-in transaction is initiated by the *MediaIn* command and remains active until the transaction is either confirmed through the Check.MediaInEnd command, or cancelled by the Check.MediaInRollback, the Check.RetractMedia or the Check.Reset command. Multiple calls to the *Check.MediaIn* command can be made while a transaction is active to obtain additional items from the customer.

The following values returned by this command can change after the media-in transaction has ended if items are later moved in the device:

- mediaOnStacker
- mediaLocation
- customerAccess

Command Message

Payload (version 2.0)
This message does not define any properties.

Completion Message

Payload (version 2.0)	Type	Required
{		
"mediaInTransaction": "active",	string	✓
"mediaOnStacker": "5",	string, null	
"lastMediaInTotal": "10",	string, null	
"lastMediaAddedToStacker": "3",	string, null	
"totalItems": "8",	string	
"totalItemsRefused": "2",	string	
"totalBunchesRefused": "1",	string	
"mediaInfo": [{	array (object), null	
"mediaID": 4,	integer	✓
"mediaLocation": "device",	string	✓
"codelineData": "□22222□□123456□",	string	
"magneticReadIndicator": "noMicr",	string	✓
"image": [{	array (object), null	
"imageSource": "back",	string	✓
"imageType": "jpg",	string	✓
"imageColorFormat": "full",	string	✓
"imageScanColor": "white",	string	✓
"imageStatus": "ok",	string	✓
"image": "wCAAAQgwMDAwMDAwMA=="	string, null	
}],		
"insertOrientation": {	object, null	

Payload (version 2.0)	Type	Required
"codeline": "top",	string, null	
"media": "down"	string, null	
},		
"mediaSize": {	object, null	
"longEdge": 205,	integer	
"shortEdge": 103	integer	
},		
"mediaValidity": "ok",	string	
"customerAccess": "customer"	string	
}]		
}		

Properties

mediaInTransaction

Status of the media-in transaction. The following values are possible:

- `ok` - The media-in transaction completed successfully.
- `active` - There is a media-in transaction active.
- `rollback` - The media-in transaction was successfully rolled back.
- `rollbackAfterDeposit` - The media-in transaction was successfully rolled back after some items had

been deposited to a storage unit. This value only applies to devices without a stacker.

- `retract` - The media-in transaction ended with the items being successfully retracted.
- `failure` - The media-in transaction failed as the result of a device failure.
- `unknown` - The state of the media-in transaction is unknown.
- `reset` - The media-in transaction ended as the result of a Reset or

CashManagement.Reset command.

mediaOnStacker

Contains the total number of media items currently on the stacker or null if the device has no stacker. This value can change outside of a transaction as the media moves within the device. Following values are possible:

- `<number>` - The number of items.
- `unknown` - The precise number of items is unknown.

Property value constraints:

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

default: null

lastMediaInTotal

Contains the number of media items processed by the last MediaIn command. This count is not modified for bunches of items which are refused as a single entity. This count only applies to devices with stackers is persistent and is therefore null if not applicable. Following values are possible:

- `<number>` - The number of items.
- `unknown` - The precise number of items is unknown.

Property value constraints:

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

default: null

Properties
lastMediaAddedToStacker Contains the number of media items on the stacker successfully accepted by the last MediaIn command. This count is persistent and is null if the device has no stacker. The number of media items refused during the last command can be determined by *lastMediaInTotal* - *lastMediaAddedToStacker*. This is only possible if these values contain values, and would not include bunches of items refused as a single entity. Following values are possible: • `<number>` - The number of items. • `unknown` - The precise number of items is unknown. Property value constraints: `pattern: ^unknown$|^[0-9]+$` default: null
totalItems The total number of items that have been allocated a media ID during the whole of the current transaction (if a transaction is active) or last transaction (if no transaction is active). This count does not include refused items and Cash items. This count is persistent. Following values are possible: • `<number>` - The number of items. • `unknown` - The precise number of items is unknown. Property value constraints: `pattern: ^unknown$|^[0-9]+$` default: "0"
totalItemsRefused Contains the total number of refused items during the execution of the whole transaction. This count does not include bunches of items which are refused as a single entity without being processed as single items. This count is persistent. Following values are possible: • `<number>` - The number of items. • `unknown` - The precise number of items is unknown. Property value constraints: `pattern: ^unknown$|^[0-9]+$` default: "0"
totalBunchesRefused Contains the total number of refused bunches of items that were not processed as single items. This count is persistent. Following values are possible: • `<number>` - The number of items. • `unknown` - The precise number of items is unknown. Property value constraints: `pattern: ^unknown$|^[0-9]+$` default: "0"
mediaInfo This array contains details of the media items processed during the current or last transaction (depending on the value of *mediaInTransaction*). The array contains one element for every item that has been allocated a media ID (i.e. items that have been reported to the application). If there are no media items then mediaInfo is null. The media info is available until a new transaction is started with the MediaIn command. The media location information may be updated after a transaction is completed, e.g. if media that was presented to the customer is subsequently retracted. The media info is persistent. default: null
mediaInfo/mediaID Specifies the sequence number (starting from 1) of a media item. Property value constraints: `minimum: 1`

Properties
mediaInfo/mediaLocation Specifies the location of the media item. This value can change outside of a media-in transaction as the media moves within the device. The following values are possible: • `device` - The media item is inside the device in some position other than a storage unit. • `<storage unit identifier>` - The media item is in a storage unit as specified by identifier. • `customer` - The media item has been returned to the customer. • `unknown` - The media item location is unknown. Property value constraints: `pattern: ^device$\|^customer$|^unknown$\|^unit[0-9A-Za-z]+$`
mediaInfo/codelineData Specifies the code line data. See Code line Characters. default: ""
mediaInfo/magneticReadIndicator Specifies the type of technology used to read a MICR code line. The following values are possible: • `micr` - The MICR code line was read using MICR technology and MICR characters were present. • `notMicr` - The MICR code line was NOT read using MICR technology. • `noMicr` - The MICR code line was read using MICR technology and no magnetic characters were read. • `unknown` - It is unknown how the MICR code line was read. • `notMicrFormat` - The code line is not a MICR format code line. • `notRead` - No code line was read.
mediaInfo/image Array of image data. If the Device has determined the orientation of the media (i.e. *insertOrientation* is defined and not set to "unknown"), then all images returned are in the standard orientation and the images will match the image source requested by the application. This means that images will be returned with the code line at the bottom, and the image of the front and rear of the media item will be returned in the structures associated with the "front" and "back" image sources respectively. default: null
mediaInfo/image/imageSource Specifies the source. The following values are possible: • `front` - The image is for the front of the media item. • `back` - The image is for the back of the media item.
mediaInfo/image/imageType Specifies the format of the image. The following values are possible: • `tif` - The image is in TIFF 6.0 format. • `wmf` - The image is in WMF (Windows Metafile) format. • `bmp` - The image is in Windows BMP format. • `jpg` - The image is in JPG format.
mediaInfo/image/imageColorFormat Specifies the color format of the image. The following values are possible: • `binary` - The image is binary (image contains two colors, usually the colors black and white). • `grayScale` - The image is gray scale (image contains multiple gray colors). • `full` - The image is full color (image contains colors like red, green, blue etc.).

Properties
mediaInfo/image/imageScanColor Selects the scan color. The following values are possible: • `red` - The image is scanned with red light. • `green` - The image is scanned with green light. • `blue` - The image is scanned with blue light. • `yellow` - The image is scanned with yellow light. • `white` - The image is scanned with white light. • `infraRed` - The image is scanned with infrared light. • `ultraViolet` - The image is scanned with ultraviolet light.
mediaInfo/image/imageStatus Status of the image data. The following values are possible: • `ok` - The data is OK. • `sourceNotSupported` - The data source or image attributes are not supported by the Service, e.g., scan color not supported. • `sourceMissing` - The image could not be obtained.

mediaInfo/image/image
Base64 encoded image. May be null if no image was obtained.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`
default: null

mediaInfo/insertOrientation This value reports how the media item was actually inserted into the input position (from the customer's perspective). The full orientation can be determined as a combination of *codeline* and *media* values. If the orientation is unknown, this will be null. default: null
mediaInfo/insertOrientation/codeline Specifies the orientation of the code line. The following values are possible, or null if unknown. • `right` - The code line is to the right. • `left` - The code line is to the left. • `bottom` - The code line is to the bottom. • `top` - The code line is to the top. default: null
mediaInfo/insertOrientation/media Specifies the orientation of the media. The following values are possible, or null if unknown: • `up` - The front of the media (the side with the code line) is facing up. • `down` - The front of the media (the side with the code line) is facing down. default: null
mediaInfo/mediaSize Specifies the size of the media item. Will be null if the device does not support media size measurement or no size measurements are known. default: null
mediaInfo/mediaSize/longEdge Specifies the length of the long edge of the media in millimeters, or 0 if unknown. Property value constraints: `minimum: 0` default: 0

Properties
mediaInfo/mediaSize/shortEdge Specifies the length of the short edge of the media in millimeters, or 0 if unknown. Property value constraints: `minimum: 0` default: 0
mediaInfo/mediaValidity Media items may have special security features which can be detected by the device. This specifies whether the media item is suspect or valid, allowing the application the choice in how to further process a media item that could not be confirmed as being valid. The following values are possible: • `ok` - The media item is valid. • `suspect` - The validity of the media item is suspect. • `unknown` - The validity of the media item is unknown. • `noValidation` - No specific security features were evaluated. default: "ok"
mediaInfo/customerAccess Specifies if the media item has been in customer access since it was first deposited, e.g. it has been retracted from a position with customer access. This value can change outside of a media-in transaction as the media moves within the device. The following values are possible: • `unknown` - It is not known if the media item has been in a position with customer access. • `customer` - The media item has been in a position with customer access. • `none` - The media item has not been in a position with customer access. default: "none"

Event Messages

None

9.2.2 Check.MediaIn

This command accepts media into the device from the input position.

A media-in transaction consists of one or more Check.MediaIn commands. A media-in transaction is initiated by the first *Check.MediaIn* command and remains active until the transaction is either confirmed through the Check.MediaInEnd command, or cancelled by the Check.MediaInRollBack, the Check.RetractMedia or the Check.Reset command. Multiple calls to the *Check.MediaIn* command can be made while a transaction is active to obtain additional items from the customer. If a media-in transaction is active (i.e. mediaInTransaction is *active*) when a *Check.MediaIn* command is successfully cancelled or times out, then the transaction remains active.

When the command is executed, if there is no media in the input slot then the device is enabled for media entry and the Check.NoMediaEvent event is generated when the device is ready to accept media. When the customer inserts the media a Check.MediaInsertedEvent event is generated and media processing begins. If media is already present at the input slot then a Check.MediaInsertedEvent event is generated and media processing begins immediately.

The Check.MediaDataEvent event delivers the code line and all requested image data during execution of this command. One event is generated for each media item scanned by this command. The Check.MediaDataEvent event is not generated for refused media items.

A failure during processing a single media item does not mean that the command has failed even if some or all of the media are refused by the media reader. In this case the command will return *success* and one or more Check.MediaRefusedEvent events will be sent to report the reasons why the items have been refused.

Refused items are not presented back to the customer with this command. The Check.MediaRefusedEvent event indicates whether or not media must be returned to the customer before further media movement commands can be executed. If the Check.MediaRefusedEvent event indicates that the media must be returned then the application must use the Check.PresentMedia command to return the refused items. If the event does not indicate that the application must return the media items then the application can still elect to return the media items using the Check.PresentMedia command or instead allow the refused items to be returned during the *Check.MediaInEnd* or *Check.MediaInRollBack* commands.

If there is no stacker on the device or applicationRefuse is true then just one of the media items inserted are processed by this command, and therefore the command completes as soon as the last image for the first item is produced or when the first item is automatically refused. If there is a stacker on the device then the command completes when the last image for the last item is produced or when the last item is refused.

Command Message

Payload (version 2.0)	Type	Required
{		
"codelineFormat": "e13b",	string, null	
"image": [{	array (object), null	
"source": "back",	string	✓
"type": "jpg",	string	✓
"colorFormat": "full",	string	✓
"scanColor": "white"	string	✓
}],		
"maxMediaOnStacker": 10,	integer	
"applicationRefuse": true	boolean	
}		

Properties
codelineFormat Specifies the code line format. May be null if no code line data is required. If supplied, it must be one of the supported code line formats. The following values are possible: • `cmc7` - Read CMC7 code line [Ref. check-4]. • `e13b` - Read E13B code line [Ref. check-3]. • `ocr` - Read code line using OCR. The default or pre-configured OCR font will be used. • `ocra` - Read code line using OCR font A [Ref. check-1]. • `ocrb` - Read code line using OCR font B [Ref. check-2]. default: null
image An array specifying the images to be read for each item. May be null if no images are required. default: null
image/source Specifies the source. The following values are possible: • `front` - The image is for the front of the media item. • `back` - The image is for the back of the media item.
image/type Specifies the format of the image. The following values are possible: • `tif` - The image is in TIFF 6.0 format. • `wmf` - The image is in WMF (Windows Metafile) format. • `bmp` - The image is in Windows BMP format. • `jpg` - The image is in JPG format.
image/colorFormat Specifies the color format of the image. The following values are possible: • `binary` - The image is binary (image contains two colors, usually the colors black and white). • `grayScale` - The image is gray scale (image contains multiple gray colors). • `full` - The image is full color (image contains colors like red, green, blue etc.).
image/scanColor Selects the scan color. The following values are possible: • `red` - The image is scanned with red light. • `green` - The image is scanned with green light. • `blue` - The image is scanned with blue light. • `yellow` - The image is scanned with yellow light. • `white` - The image is scanned with white light. • `infraRed` - The image is scanned with infrared light. • `ultraViolet` - The image is scanned with ultraviolet light.
maxMediaOnStacker Maximum number of media items allowed on the stacker during the media-in transaction. This value is used to limit the total number of media items on the stacker. When this limit is reached all further media items will be refused and a Check.MediaRefusedEvent message will be generated reporting *stackerFull*. • This value cannot exceed maxMediaOnStacker or the Service will return a *invalidData* error. • If 0 then the maximum number of items allowed on the stacker reported in maxMediaOnStacker will be used. • Ignored unless specified on the first Check.MediaIn command within a single media-in transaction. • Ignored on devices without stackers. Property value constraints: `minimum: 0` default: 0

Properties
applicationRefuse Specifies if the application wants to make the decision to accept or refuse each media item that has successfully been accepted by the device. • If true then the application must decide to accept or refuse each item. The application must use the Check.AcceptItem and Check.GetNextItem commands in a sequential manner to process the bunch of media inserted during the Check.MediaIn command. • If false then any decision on whether an item should be refused is left to the device/Service. • Ignored unless specified on the first Check.MediaIn command within a single media-in transaction. • Ignored if applicationRefuse is false. default: false

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "mediaRejected",	string, null	
"mediaIn": {	object, null	
"mediaOnStacker": 10,	integer, null	
"lastMedia": 5,	integer, null	
"lastMediaOnStacker": 3,	integer, null	
"mediaFeeder": "notEmpty"	string, null	
}		
}		

Properties
errorCode Specifies the error code if applicable, otherwise null. Following values are possible: • `stackerFull` - The internal stacker is already full or has already reached the limit specified as an input parameter. No media items can be accepted. • `shutterFail` - Open or close of the shutter failed due to manipulation or hardware error. • `mediaJammed` - The media is jammed. • `refusedItems` - Programming error: refused items that must be returned via the Check.PresentMedia command have not been presented (see presentRequired). • `allBinsFull` - All storage units are unusable due to being full, missing or inoperative, so no further items can be accepted. • `scannerInop` - Only images were requested by the application and these cannot be obtained because the image scanner is inoperative. • `micrInop` - Only MICR data was requested by the application and it cannot be obtained because the MICR reader is inoperative. • `positionNotEmpty` - One of the input/output/refused positions is not empty and items cannot be inserted until the media items in the position are removed. • `feederNotEmpty` - The media feeder is not empty. This only applies when the Check.GetNextItem command should be used to retrieve the next media item. • `mediaRejected` - The media was rejected before it was fully inserted within the device. The Check.MediaRejectedEvent is posted with the details. The device is still operational. • `feederInop` - The media feeder is inoperative. • `mediaPresent` - Media from a previous transaction is present in the device when an attempt to start a new media-in transaction was made. The media must be cleared before a new transaction can be started. default: null

Properties
mediaIn Describes the outcome of the command, defining where all the media was moved. May be null if no media items were moved. default: null
mediaIn/mediaOnStacker Contains the total number of media items on the stacker (including *lastMediaOnStacker*). May be null if it is unknown or the device does not have a stacker. Property value constraints: `minimum: 0` default: null
mediaIn/lastMedia Contains the number of media items processed by this instance of the command execution. May be null if it is unknown or the device does not have a stacker. Property value constraints: `minimum: 0` default: null
mediaIn/lastMediaOnStacker Contains the number of media items on the stacker successfully accepted by this instance of the command execution. May be null if it is unknown or the device does not have a stacker. The number of refused media items can be determined by *lastMedia* - *lastMediaOnStacker*. This is only possible if these values contain known values, and would not be possible if a bunch of items were refused as a single entity. Property value constraints: `minimum: 0` default: null
mediaIn/mediaFeeder Supplies the state of the media feeder. This value indicates if there are items on the media feeder waiting for processing via the Check.GetNextItem command. If null, the device has no media feeder or the capability to report the status of the media feeder is not supported by the device. This value can be one of the following values: • `empty` - The media feeder is empty. • `notEmpty` - The media feeder is not empty. • `inoperative` - The media feeder is inoperative. • `unknown` - Due to a hardware error or other condition, the state of the media feeder cannot be determined. default: null

Event Messages

- Check.NoMediaEvent
- Check.MediaInsertedEvent
- Check.MediaRefusedEvent
- Check.MediaDataEvent
- Check.MediaRejectedEvent

9.2.3 Check.MediaInEnd

This command ends a media-in transaction. If media items are on the stacker as a result of a Check.MediaIn command, they are moved to the destination specified by Check.SetMediaParameters. Any additional actions specified for the items by *Check.SetMediaParameters* such as printing, stamping and rescanning are also executed. If the destination has not been set for a media item then the Service will decide which storage unit to put the item into. If no items are in the device the command will complete with the *noMediaPresent* error and mediaInTransaction will be set to *ok*.

The way in which media is returned to the customer as a result of this command is defined by presentControl. If false, the application must call Check.PresentMedia to present the media items to be returned as a result of this command. If true the Service presents any returned items implicitly and the application does not need to call *Check.PresentMedia*.

If items have been refused and the Check.MediaRefusedEvent message has indicated that the items must be returned (i.e. presentRequired is true) then these items must be returned using the *Check.PresentMedia* command before the Check.MediaInEnd command is issued, otherwise a *refusedItems* error will be returned. If items have been refused and the *Check.MediaRefusedEvent* event has indicated that the items do not need to be returned (i.e. *presentRequired* is false) then the *Check.MediaInEnd* command causes any refused items which have not yet been returned to the customer (via the *Check.PresentMedia* command) to be returned along with any items that the application has selected to return to the customer (via the *Check.SetMediaParameters* command). Even if all items are being deposited, previously refused items will be returned to the customer by this command. The Check.MediaPresentedEvent event(s) inform the application of the position where the media has been presented to.

This command completes when all the media items have been put into their specified storage units and in the case where media is returned to the customer as a result of this command, after the last bunch of media items to be returned to the customer has been presented, but before the last bunch is taken.

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

Command Message

Payload (version 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "refusedItems",`	string, null	
`"mediaInEnd": {`	object, null	
`"itemsReturned": 2,`	integer	
`"itemsRefused": 3,`	integer	
`"bunchesRefused": 1,`	integer	
`"storage": {`	object, null	
`"storage": {`	object, null	
`"unit1": {`	object	
`"id": "RC1",`	string, null	
`"positionName": "Top Right",`	string, null	
`"capacity": 100,`	integer, null	
`"status": "ok",`	string, null	
`"serialNumber": "ABCD1234",`	string, null	
`"cash": {`	object, null	

Payload (version 2.0)	Type	Required
"capabilities": {	object, null	
"types": {	object, null	
"cashIn": true,	boolean, null	
"cashOut": false,	boolean, null	
"replenishment": false,	boolean, null	
"cashInRetract": false,	boolean, null	
"cashOutRetract": false,	boolean, null	
"reject": false	boolean, null	
},		
"items": {	object, null	
"fit": false,	boolean, null	
"unfit": false,	boolean, null	
"unrecognized": false,	boolean, null	
"counterfeit": false,	boolean, null	
"suspect": false,	boolean, null	
"inked": false,	boolean, null	
"coupon": false,	boolean, null	
"document": false	boolean, null	
},		
"hardwareSensors": false,	boolean, null	
"retractAreas": 1,	integer, null	
"retractThresholds": false,	boolean, null	
"cashItems": ["type20USD1", "type50USD1"]	array (string), null	
},		
"configuration": {	object, null	
"types": See mediaInEnd/storage/storage/unit1/cash/capabilities/types properties	object, null	

Payload (version 2.0)	Type	Required
"items": See mediaInEnd/storage/storage/unit1/cash/capabilities/items properties	object, null	
"currency": "USD",	string, null	
"value": 20.00,	number, null	
"highThreshold": 500,	integer, null	
"lowThreshold": 10,	integer, null	
"appLockIn": false,	boolean, null	
"appLockOut": false,	boolean, null	
"cashItems": See mediaInEnd/storage/storage/unit1/cash/capabilities/cashItems,	array (string), null	
"name": "$10",	string, null	
"maxRetracts": 5	integer, null	
},		
"status": {	object, null	
"index": 4,	integer	✓
"initial": {	object, null	
"unrecognized": 5,	integer, null	
"*type20USD1*": {	object, null	
"fit": 15,	integer, null	
"unfit": 0,	integer, null	
"suspect": 0,	integer, null	
"counterfeit": 0,	integer, null	
"inked": 0	integer, null	
},		
"*type50USD1*": See *mediaInEnd/storage/storage/unit1/cash/status/initial/type20USD1* properties	object, null	
},		
"out": {	object, null	
"presented": See mediaInEnd/storage/storage/unit1/cash/status/initial properties	object, null	
"rejected": See mediaInEnd/storage/storage/unit1/cash/status/initial properties	object, null	

Payload (version 2.0)	Type	Required
"distributed": See mediaInEnd/storage/storage/unit1/cash/status/initial properties	object, null	
"unknown": See mediaInEnd/storage/storage/unit1/cash/status/initial properties	object, null	
"stacked": See mediaInEnd/storage/storage/unit1/cash/status/initial properties	object, null	
"diverted": See mediaInEnd/storage/storage/unit1/cash/status/initial properties	object, null	
"transport": See mediaInEnd/storage/storage/unit1/cash/status/initial properties	object, null	
},		
"in": {	object, null	
"retractOperations": 15,	integer, null	
"deposited": See mediaInEnd/storage/storage/unit1/cash/status/initial properties	object, null	
"retracted": See mediaInEnd/storage/storage/unit1/cash/status/initial properties	object, null	
"rejected": See mediaInEnd/storage/storage/unit1/cash/status/initial properties	object, null	
"distributed": See mediaInEnd/storage/storage/unit1/cash/status/initial properties	object, null	
"transport": See mediaInEnd/storage/storage/unit1/cash/status/initial properties	object, null	
},		
"accuracy": "accurate",	string, null	
"replenishmentStatus": "ok",	string, null	
"operationStatus": "dispenseInoperative"	string, null	
}		
},		
"card": {	object, null	
"capabilities": {	object, null	
"type": "retain",	string, null	
"hardwareSensors": true	boolean, null	
},		
"configuration": {	object, null	
"cardID": "LoyaltyCard",	string, null	
"threshold": 10	integer, null	
},		
"status": {	object, null	
"initialCount": 0,	integer, null	
"count": 0,	integer, null	

Payload (version 2.0)	Type	Required
"retainCount": 0,	integer, null	
"replenishmentStatus": "ok"	string, null	
}		
},		
"check": {	object, null	
"capabilities": {	object, null	
"types": {	object, null	
"mediaIn": true,	boolean, null	
"retract": false	boolean, null	
},		
"sensors": {	object, null	
"empty": false,	boolean, null	
"high": false,	boolean, null	
"full": false	boolean, null	
}		
},		
"configuration": {	object, null	
"types": See mediaInEnd/storage/storage/unit1/check/capabilities/types properties	object, null	
"binID": "My check bin",	string, null	
"highThreshold": 500,	integer, null	
"retractHighThreshold": 5	integer, null	
},		
"status": {	object, null	
"index": 4,	integer	✓
"initial": {	object, null	
"mediaInCount": 100,	integer, null	
"count": 150,	integer, null	
"retractOperations": 15	integer, null	
},		
"in": See mediaInEnd/storage/storage/unit1/check/status/initial properties	object, null	
"replenishmentStatus": "high"	string, null	

Payload (version 2.0)	Type	Required
} } }, "*unit2*": See *mediaInEnd/storage/storage/unit1* properties } } } }	object	

Properties

errorCode

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

- `noMedia` - No media is present in the device.
- `shutterFail` - Open or close of the shutter failed due to manipulation or hardware error.
- `mediaJammed` - The media is jammed.
- `mediaBinError` - A storage unit caused a problem. A Storage.StorageErrorEvent will be posted with the details.
- `positionNotEmpty` - One of the input/output/refused positions is not empty and items cannot be inserted until the media items in the position are removed.
- `refusedItems` - Programming error: refused items that must be returned via the Check.PresentMedia command have not been presented (see presentRequired).
- `feederNotEmpty` - The media feeder is not empty.

default: null

mediaInEnd

Describes the outcome of the command and hence the outcome of the transaction, defining where all the media items were moved.

default: null

mediaInEnd/itemsReturned

Contains the number of media items that were returned to the customer by application selection through the Check.SetMediaParameters command during the current transaction. This does not include items that were refused.

Property value constraints:

```
minimum: 0
```

default: 0

mediaInEnd/itemsRefused

Contains the total number of items automatically returned to the customer during the execution of the whole transaction. This count does not include bunches of items which are refused as a single entity without being processed as single items.

Property value constraints:

```
minimum: 0
```

default: 0

mediaInEnd/bunchesRefused

Contains the total number of refused bunches of items that were automatically returned to the customer without being processed as single items.

Property value constraints:

```
minimum: 0
```

default: 0

Properties
mediaInEnd/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. default: null
mediaInEnd/storage/storage Object containing storage unit information. The property name is the storage unit identifier. default: null
mediaInEnd/storage/storage/unit1 (example name) The object contains a single storage unit. Property name constraints: `pattern: ^unit[0-9A-Za-z]+$`
mediaInEnd/storage/storage/unit1/id An identifier which can be used for cUnitID in CDM/CIM XFS 3.x migration. May be null if not applicable. Property value constraints: `pattern: ^.{1,5}$` default: null
mediaInEnd/storage/storage/unit1/positionName Fixed physical name for the position. May be null if not applicable. default: null
mediaInEnd/storage/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, null means capacity is not applicable. Property value constraints: `minimum: 0` default: null
mediaInEnd/storage/storage/unit1/status The state of the unit. This property may be null in events if the state did not change, otherwise 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. default: null
mediaInEnd/storage/storage/unit1/serialNumber The storage unit's serial number if it can be read electronically. May be null if not applicable. default: null
mediaInEnd/storage/storage/unit1/cash The cash related contents, status and configuration of the unit. May be null if not applicable. default: null
mediaInEnd/storage/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. This may be null in events if capabilities have not changed. default: null

Properties
mediaInEnd/storage/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. May be null in command data or events if not changed or being changed. default: null
mediaInEnd/storage/storage/unit1/cash/capabilities/types/cashIn The unit can accept cash items. If *cashOut* is also true then the unit can recycle. May be null in command data or events if not changed or being changed. default: null
mediaInEnd/storage/storage/unit1/cash/capabilities/types/cashOut The unit can dispense cash items. If *cashIn* is also true then the unit can recycle. May be null in command data or events if not changed or being changed. default: null
mediaInEnd/storage/storage/unit1/cash/capabilities/types/replenishment Replenishment container. A storage unit can be refilled from or emptied to a replenishment container. May be null in command data or events if not changed or being changed. default: null
mediaInEnd/storage/storage/unit1/cash/capabilities/types/cashInRetract Retract unit. Items can be retracted into this unit during Cash In operations. May be null in command data or events if not changed or being changed. default: null
mediaInEnd/storage/storage/unit1/cash/capabilities/types/cashOutRetract Retract unit. Items can be retracted into this unit during Cash Out operations. May be null in command data or events if not changed or being changed. default: null
mediaInEnd/storage/storage/unit1/cash/capabilities/types/reject Reject unit. Items can be rejected into this unit. May be null in command data or events if not changed or being changed. default: null
mediaInEnd/storage/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. May be null in command data if not being changed. May be null in command data or events if not changed or being changed. default: null
mediaInEnd/storage/storage/unit1/cash/capabilities/items/fit The storage unit can store cash items which are fit for recycling. May be null in command data or events if not changed or being changed. default: null
mediaInEnd/storage/storage/unit1/cash/capabilities/items/unfit The storage unit can store cash items which are unfit for recycling. May be null in command data or events if not changed or being changed. default: false
mediaInEnd/storage/storage/unit1/cash/capabilities/items/unrecognized The storage unit can store unrecognized cash items. May be null in command data or events if not changed or being changed. default: null

Properties
mediaInEnd/storage/storage/unit1/cash/capabilities/items/counterfeit The storage unit can store counterfeit cash items. May be null in command data or events if not changed or being changed. default: null
mediaInEnd/storage/storage/unit1/cash/capabilities/items/suspect The storage unit can store suspect counterfeit cash items. May be null in command data or events if not changed or being changed. default: null
mediaInEnd/storage/storage/unit1/cash/capabilities/items/inked The storage unit can store cash items which have been identified as ink stained. May be null in command data or events if not changed or being changed. default: null
mediaInEnd/storage/storage/unit1/cash/capabilities/items/coupon Storage unit containing coupons or advertising material. May be null in command data or events if not changed or being changed. default: null
mediaInEnd/storage/storage/unit1/cash/capabilities/items/document Storage unit containing documents. May be null in command data or events if not changed or being changed. default: null
mediaInEnd/storage/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. May be null in command data or events if not changed or being changed. default: null
mediaInEnd/storage/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. May be null if items can not be retracted into this storage unit or in events if not changed or being changed. Property value constraints: `minimum: 1` default: null
mediaInEnd/storage/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. May be null if items can not be retracted into this storage unit or in events if not changed or being changed. default: null
mediaInEnd/storage/storage/unit1/cash/capabilities/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. May be null in command data or events if not being modified. Property value constraints: `minItems: 1` default: null

Properties
mediaInEnd/storage/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. May be null in command data or events if no configuration is to be or has been changed. default: null
mediaInEnd/storage/storage/unit1/cash/configuration/currency ISO 4217 currency identifier [Ref. cashmanagement-1]. May only be modified in an exchange state if applicable. May be null if the unit is configured to store mixed currencies or non-cash items. Property value constraints: `pattern: ^[A-Z]{3}$` default: null
mediaInEnd/storage/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. May be null in command data or events if not being modified. Property value constraints: `minimum: 0` default: null
mediaInEnd/storage/storage/unit1/cash/configuration/highThreshold If specified, replenishmentStatus is 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 null, high is based on hardware sensors if supported - see hardwareSensors. May be null in command data or events if not being modified. Property value constraints: `minimum: 1` default: null
mediaInEnd/storage/storage/unit1/cash/configuration/lowThreshold If specified, replenishmentStatus is 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 null, low is based on hardware sensors if supported - see hardwareSensors. May be null in command data or events if not being modified. Property value constraints: `minimum: 1` default: null
mediaInEnd/storage/storage/unit1/cash/configuration/appLockIn If true, items cannot be accepted into the storage unit in Cash In operations. May be null in command data or events if not being modified. default: null
mediaInEnd/storage/storage/unit1/cash/configuration/appLockOut If true, items cannot be dispensed from the storage unit in Cash Out operations. May be null in command data or events if not being modified. default: null
mediaInEnd/storage/storage/unit1/cash/configuration/name Application configured name of the unit. May be null in command data or events if not being modified. default: null

Properties
mediaInEnd/storage/storage/unit1/cash/configuration/maxRetracts If specified, this is the number of retract operations allowed into the unit. Only applies to retract units. If retractOperations equals this number, then no further retracts are allowed into this storage unit. If null in output, the maximum number is not limited by counts. May be null in command data or events if not being modified. Property value constraints: `minimum: 1` default: null
mediaInEnd/storage/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. May be null in events if not changing. default: null
mediaInEnd/storage/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` default: null
mediaInEnd/storage/storage/unit1/cash/status/initial The cash related items which were in the storage unit at the last replenishment. default: null
mediaInEnd/storage/storage/unit1/cash/status/initial/unrecognized Count of unrecognized items handled by the cash interface. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
mediaInEnd/storage/storage/unit1/cash/status/initial/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification. default: null
mediaInEnd/storage/storage/unit1/cash/status/initial/type20USD1/fit Count of genuine cash items which are fit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
mediaInEnd/storage/storage/unit1/cash/status/initial/type20USD1/unfit Count of genuine cash items which are unfit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
mediaInEnd/storage/storage/unit1/cash/status/initial/type20USD1/suspect Count of suspected counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null

Properties
mediaInEnd/storage/storage/unit1/cash/status/initial/type20USD1/counterfeit Count of counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
mediaInEnd/storage/storage/unit1/cash/status/initial/type20USD1/inked Count of cash items which have been identified as ink stained. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
mediaInEnd/storage/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. May be null if items have not or can not be moved from the storage unit by cash commands. default: null
mediaInEnd/storage/storage/unit1/cash/status/out/presented The items dispensed from this storage unit which are or were customer accessible. Will be null if no items were presented. default: null
mediaInEnd/storage/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. Will be null if no items were rejected. default: null
mediaInEnd/storage/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. Will be null if no items were distributed. default: null
mediaInEnd/storage/storage/unit1/cash/status/out/unknown The items dispensed from this storage unit which moved to an unknown position. Will be null if no items were unknown. default: null
mediaInEnd/storage/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. Will be null if no items were stacked. default: null
mediaInEnd/storage/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. Will be null if no items were diverted. default: null

Properties
mediaInEnd/storage/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. Will be null if no items apply. default: null
mediaInEnd/storage/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. May be null if items have not or can not be moved into the storage unit by cash commands. default: null
mediaInEnd/storage/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. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
mediaInEnd/storage/storage/unit1/cash/status/in/deposited The items deposited in the storage unit during a Cash In transaction. Can be null, if all values are 0. default: null
mediaInEnd/storage/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*. Can be null, if all values are 0. default: null
mediaInEnd/storage/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. Can be null, if all values are 0. default: null
mediaInEnd/storage/storage/unit1/cash/status/in/distributed The items deposited in this storage unit originating from another storage unit but not rejected. Can be null, if all values are 0. default: null
mediaInEnd/storage/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. Can be null, if all values are 0. default: null

Properties
mediaInEnd/storage/storage/unit1/cash/status/accuracy Describes the accuracy of the counts reported by *out* and *in*. If null in Storage.GetStorage, the hardware is not capable of determining the accuracy, otherwise the following values are possible: • `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. default: null
mediaInEnd/storage/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. May be null in events if not changing, otherwise 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. default: null
mediaInEnd/storage/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 null in Storage.GetStorage, *status* and *replenishmentStatus* apply to both cash out and cash in operations. default: null
mediaInEnd/storage/storage/unit1/card The card related contents, status and configuration of the unit. May be null if not applicable. default: null
mediaInEnd/storage/storage/unit1/card/capabilities Indicates the card storage unit capabilities. This property can be null if a change is being reported using Storage.StorageChangedEvent or Storage.StorageThresholdEvent.

Properties
mediaInEnd/storage/storage/unit1/card/capabilities/type The type of card storage. This property may be null in events if the type did not change, otherwise will be one of the following values: • `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. default: null
mediaInEnd/storage/storage/unit1/card/capabilities/hardwareSensors Indicates whether the storage unit has hardware sensors that can detect threshold states. This property may be null in events if it did not change. default: null
mediaInEnd/storage/storage/unit1/card/configuration Indicates the card storage unit configuration. This property can be null if the storage unit is being set using Storage.SetStorage, or a change is being reported using Storage.StorageChangedEvent or Storage.StorageThresholdEvent.
mediaInEnd/storage/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 storage units and may be null in events if it did not change. default: null
mediaInEnd/storage/storage/unit1/card/configuration/threshold If the threshold value is non zero, hardware sensors in the storage unit do not trigger Storage.StorageThresholdEvent events. This property may be null in events if it did not change. 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` default: null
mediaInEnd/storage/storage/unit1/card/status Indicates the card storage unit status. This property can be null if a change is being reported using Storage.StorageChangedEvent or Storage.StorageThresholdEvent.
mediaInEnd/storage/storage/unit1/card/status/initialCount The initial number of cards in the storage unit. This is only applicable to dispense type storage units. This property may be null in events if it did not change. This value is persistent. Property value constraints: `minimum: 0` default: null

Properties
mediaInEnd/storage/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 and may be null in events if it did not change. Property value constraints: `minimum: 0` default: null
mediaInEnd/storage/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 and may be null in events if it did not change. Property value constraints: `minimum: 0` default: null
mediaInEnd/storage/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 null. The property may also be null in events if it did not change. 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. default: null
mediaInEnd/storage/storage/unit1/check The check related contents, status and configuration of the unit. May be null if not applicable. default: null
mediaInEnd/storage/storage/unit1/check/capabilities Indicates what the storage unit is capable of - this includes information which is a combination of that reported in WFS_INF_IPM_MEDIA_BIN_INFO and WFS_INF_IPM_MEDIA_BIN_CAPABILITIES in XFS 3.x. May be null in events if not changed. default: null
mediaInEnd/storage/storage/unit1/check/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. May be null in command data and events if not changing. default: null

Properties
mediaInEnd/storage/storage/unit1/check/capabilities/types/mediaIn The unit can accept items during Media In transactions. May be null in command data and events if not changing. default: null
mediaInEnd/storage/storage/unit1/check/capabilities/types/retract Retract unit. Items can be retracted into this unit using Check.RetractMedia. May be null in command data and events if not changing. default: null
mediaInEnd/storage/storage/unit1/check/capabilities/sensors The types of sensor the unit has. May be null in command data and events if not changing. default: null
mediaInEnd/storage/storage/unit1/check/capabilities/sensors/empty The unit contains a hardware sensor which reports when the unit is empty. May be null in command data and events if not changing. default: null
mediaInEnd/storage/storage/unit1/check/capabilities/sensors/high The unit contains a hardware sensor which reports when the unit is nearly full. May be null in command data and events if not changing. default: null
mediaInEnd/storage/storage/unit1/check/capabilities/sensors/full The unit contains a hardware sensor which reports when the unit is full. May be null in command data and events if not changing. default: null
mediaInEnd/storage/storage/unit1/check/configuration Indicates what the storage unit is configured to do - where applicable the supported options can be derived from capabilities. May be null in command data and events if not being modified.
mediaInEnd/storage/storage/unit1/check/configuration/binID An application defined Storage Unit Identifier. This may be null in events if not changing. default: null
mediaInEnd/storage/storage/unit1/check/configuration/highThreshold If specified, replenishmentStatus is set to *high* if the total number of items in the storage unit is greater than this number. May be null in command data and events if not being modified. Property value constraints: `minimum: 1` default: null
mediaInEnd/storage/storage/unit1/check/configuration/retractHighThreshold If specified and the storage unit is configured as *retract*, replenishmentStatus is set to *high* if the total number of retract operations in the storage unit is greater than this number. May be null in command data and events if not being modified. Property value constraints: `minimum: 0` default: null
mediaInEnd/storage/storage/unit1/check/status Indicates the storage unit status. May be null in events where status has not changed. default: null

Properties
mediaInEnd/storage/storage/unit1/check/status/index Assigned by the Service. Will be a unique number which can be used to determine *usBinNumber* in XFS 3.x migration. This can change as storage units are added and removed from the storage collection. Property value constraints: `minimum: 1`
mediaInEnd/storage/storage/unit1/check/status/initial The check related counts as set at the last replenishment. May be null in events where status has not changed. default: null
mediaInEnd/storage/storage/unit1/check/status/initial/mediaInCount Count of items added to the storage unit due to Check operations. If the number of items is not counted this is not reported and *retractOperations* is incremented as items are added to the unit. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
mediaInEnd/storage/storage/unit1/check/status/initial/count Total number of items added to the storage unit due to any operations. If the number of items is not counted this is not reported and *retractOperations* is incremented as items are added to the unit. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
mediaInEnd/storage/storage/unit1/check/status/initial/retractOperations Total number of operations which resulted in items being retracted to the storage unit. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
mediaInEnd/storage/storage/unit1/check/status/in The check items added to the unit since the last replenishment. May be null in events where status has not changed. default: null
mediaInEnd/storage/storage/unit1/check/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. May be null in command data and events if not changing, otherwise 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 a full sensor or counts. • `high` - The storage unit is almost full (either high sensor based or exceeded the highThreshold or retractHighThreshold). • `empty` - The storage unit is empty, or insufficient items in the storage unit are preventing further dispense operations. If the storage unit has the empty sensor, this state is not set by counts. default: null

Event Messages

- Check.MediaDataEvent
- Storage.StorageErrorEvent
- Check.MediaPresentedEvent

9.2.4 Check.MediaInRollback

This command ends a media-in transaction. All media that is in the device as a result of Check.MediaIn commands is returned to the customer. Nothing is printed on the media. If no items are in the device the command will complete with the *noMediaPresent* error and mediaInTransaction will be set to *rollback*.

The way in which media is returned to the customer as a result of this command is defined by presentControl. If false, the application must call Check.PresentMedia to present the media items to be returned as a result of this command. If true the Service presents any returned items implicitly and the application does not need to call Check.PresentMedia.

If items have been refused and the Check.MediaRefusedEvent message has indicated that the items must be returned (i.e. presentRequired is true) then these items must be returned using the Check.PresentMedia command before the Check.MediaInRollBack command is issued, otherwise a *refusedItems* error will be returned. If items have been refused and the *Check.MediaRefusedEvent* has indicated that the items do not need to be returned (i.e. *presentRequired* is false) then the *Check.MediaInRollBack* command causes any refused items which have not yet been returned to the customer (via the Check.PresentMedia command) to be returned along with any items that are returned as a result of the rollback. The Check.MediaPresentedEvent event(s) inform the application of the position where the media has been presented to.

In the case where media is returned to the customer as a result of this command, this command completes when the last bunch of media items to be returned to the customer has been presented, but before the last bunch is taken.

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

Command Message

Payload (version 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "mediaJammed",`	string, null	
`"mediaInRollback": {`	object, null	
`"itemsReturned": 2,`	integer	
`"itemsRefused": 3,`	integer	
`"bunchesRefused": 1,`	integer	
`"storage": {`	object, null	
`"storage": {`	object, null	
`"unit1": {`	object	
`"id": "RC1",`	string, null	
`"positionName": "Top Right",`	string, null	
`"capacity": 100,`	integer, null	
`"status": "ok",`	string, null	
`"serialNumber": "ABCD1234",`	string, null	
`"cash": {`	object, null	

Payload (version 2.0)	Type	Required
"capabilities": {	object, null	
"types": {	object, null	
"cashIn": true,	boolean, null	
"cashOut": false,	boolean, null	
"replenishment": false,	boolean, null	
"cashInRetract": false,	boolean, null	
"cashOutRetract": false,	boolean, null	
"reject": false	boolean, null	
},		
"items": {	object, null	
"fit": false,	boolean, null	
"unfit": false,	boolean, null	
"unrecognized": false,	boolean, null	
"counterfeit": false,	boolean, null	
"suspect": false,	boolean, null	
"inked": false,	boolean, null	
"coupon": false,	boolean, null	
"document": false	boolean, null	
},		
"hardwareSensors": false,	boolean, null	
"retractAreas": 1,	integer, null	
"retractThresholds": false,	boolean, null	
"cashItems": ["type20USD1", "type50USD1"]	array (string), null	
},		

Payload (version 2.0)	Type	Required
"configuration": {	object, null	
"types": See mediaInRollback/storage/storage/unit1/cash/capabilities/types properties	object, null	
"items": See mediaInRollback/storage/storage/unit1/cash/capabilities/items properties	object, null	
"currency": "USD",	string, null	
"value": 20.00,	number, null	
"highThreshold": 500,	integer, null	
"lowThreshold": 10,	integer, null	
"appLockIn": false,	boolean, null	
"appLockOut": false,	boolean, null	
"cashItems": See mediaInRollback/storage/storage/unit1/cash/capabilities/cashItems,	array (string), null	
"name": "$10",	string, null	
"maxRetracts": 5	integer, null	
},		
"status": {	object, null	
"index": 4,	integer	✓
"initial": {	object, null	
"unrecognized": 5,	integer, null	
"*type20USD1*": {	object, null	
"fit": 15,	integer, null	
"unfit": 0,	integer, null	
"suspect": 0,	integer, null	
"counterfeit": 0,	integer, null	
"inked": 0	integer, null	
},		

Payload (version 2.0)	Type	Required
"*type50USD1*": See *mediaInRollback/storage/storage/unit1/cash/status/initial/type20USD1* properties	object, null	
},		
"out": {	object, null	
"presented": See mediaInRollback/storage/storage/unit1/cash/status/initial properties	object, null	
"rejected": See mediaInRollback/storage/storage/unit1/cash/status/initial properties	object, null	
"distributed": See mediaInRollback/storage/storage/unit1/cash/status/initial properties	object, null	
"unknown": See mediaInRollback/storage/storage/unit1/cash/status/initial properties	object, null	
"stacked": See mediaInRollback/storage/storage/unit1/cash/status/initial properties	object, null	
"diverted": See mediaInRollback/storage/storage/unit1/cash/status/initial properties	object, null	
"transport": See mediaInRollback/storage/storage/unit1/cash/status/initial properties	object, null	
},		
"in": {	object, null	
"retractOperations": 15,	integer, null	
"deposited": See mediaInRollback/storage/storage/unit1/cash/status/initial properties	object, null	
"retracted": See mediaInRollback/storage/storage/unit1/cash/status/initial properties	object, null	
"rejected": See mediaInRollback/storage/storage/unit1/cash/status/initial properties	object, null	
"distributed": See mediaInRollback/storage/storage/unit1/cash/status/initial properties	object, null	
"transport": See mediaInRollback/storage/storage/unit1/cash/status/initial properties	object, null	
},		
"accuracy": "accurate",	string, null	
"replenishmentStatus": "ok",	string, null	
"operationStatus": "dispenseInoperative"	string, null	
}		

Payload (version 2.0)	Type	Required
},		
"card": {	object, null	
"capabilities": {	object, null	
"type": "retain",	string, null	
"hardwareSensors": true	boolean, null	
},		
"configuration": {	object, null	
"cardID": "LoyaltyCard",	string, null	
"threshold": 10	integer, null	
},		
"status": {	object, null	
"initialCount": 0,	integer, null	
"count": 0,	integer, null	
"retainCount": 0,	integer, null	
"replenishmentStatus": "ok"	string, null	
}		
},		
"check": {	object, null	
"capabilities": {	object, null	
"types": {	object, null	
"mediaIn": true,	boolean, null	
"retract": false	boolean, null	
},		
"sensors": {	object, null	
"empty": false,	boolean, null	
"high": false,	boolean, null	
"full": false	boolean, null	
}		

Payload (version 2.0)	Type	Required
},		
"configuration": {	object, null	
"types": See mediaInRollback/storage/storage/unit1/check/capabilities/types properties	object, null	
"binID": "My check bin",	string, null	
"highThreshold": 500,	integer, null	
"retractHighThreshold": 5	integer, null	
},		
"status": {	object, null	
"index": 4,	integer	✓
"initial": {	object, null	
"mediaInCount": 100,	integer, null	
"count": 150,	integer, null	
"retractOperations": 15	integer, null	
},		
"in": See mediaInRollback/storage/storage/unit1/check/status/initial properties	object, null	
"replenishmentStatus": "high"	string, null	
}		
}		
},		
"*unit2*": See *mediaInRollback/storage/storage/unit1* properties	object	
}		
}		
}		
}		

Properties
errorCode Specifies the error code if applicable, otherwise null. Following values are possible: • `noMedia` - No media is present in the device. • `shutterFail` - Open or close of the shutter failed due to manipulation or hardware error. • `mediaJammed` - The media is jammed. • `positionNotEmpty` - One of the input/output/refused positions is not empty and items cannot be inserted until the media items in the position are removed. • `refusedItems` - Programming error: refused items that must be returned via the Check.PresentMedia command have not been presented (see presentRequired). default: null
mediaInRollback Describes the outcome of the command and hence the outcome of the transaction, defining where all the media items were moved. default: null
mediaInRollback/itemsReturned Contains the number of media items that were returned to the customer by application selection through the Check.SetMediaParameters command during the current transaction. This does not include items that were refused. Property value constraints: `minimum: 0` default: 0
mediaInRollback/itemsRefused Contains the total number of items automatically returned to the customer during the execution of the whole transaction. This count does not include bunches of items which are refused as a single entity without being processed as single items. Property value constraints: `minimum: 0` default: 0
mediaInRollback/bunchesRefused Contains the total number of refused bunches of items that were automatically returned to the customer without being processed as single items. Property value constraints: `minimum: 0` default: 0
mediaInRollback/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. default: null
mediaInRollback/storage/storage Object containing storage unit information. The property name is the storage unit identifier. default: null
mediaInRollback/storage/storage/unit1 (example name) The object contains a single storage unit. Property name constraints: `pattern: ^unit[0-9A-Za-z]+$`
mediaInRollback/storage/storage/unit1/id An identifier which can be used for cUnitID in CDM/CIM XFS 3.x migration. May be null if not applicable. Property value constraints: `pattern: ^.{1,5}$` default: null

Properties
mediaInRollback/storage/storage/unit1/positionName Fixed physical name for the position. May be null if not applicable. default: null
mediaInRollback/storage/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, null means capacity is not applicable. Property value constraints: `minimum: 0` default: null
mediaInRollback/storage/storage/unit1/status The state of the unit. This property may be null in events if the state did not change, otherwise 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. default: null
mediaInRollback/storage/storage/unit1/serialNumber The storage unit's serial number if it can be read electronically. May be null if not applicable. default: null
mediaInRollback/storage/storage/unit1/cash The cash related contents, status and configuration of the unit. May be null if not applicable. default: null
mediaInRollback/storage/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. This may be null in events if capabilities have not changed. default: null
mediaInRollback/storage/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. May be null in command data or events if not changed or being changed. default: null
mediaInRollback/storage/storage/unit1/cash/capabilities/types/cashIn The unit can accept cash items. If *cashOut* is also true then the unit can recycle. May be null in command data or events if not changed or being changed. default: null
mediaInRollback/storage/storage/unit1/cash/capabilities/types/cashOut The unit can dispense cash items. If *cashIn* is also true then the unit can recycle. May be null in command data or events if not changed or being changed. default: null
mediaInRollback/storage/storage/unit1/cash/capabilities/types/replenishment Replenishment container. A storage unit can be refilled from or emptied to a replenishment container. May be null in command data or events if not changed or being changed. default: null

Properties
mediaInRollback/storage/storage/unit1/cash/capabilities/types/cashInRetract Retract unit. Items can be retracted into this unit during Cash In operations. May be null in command data or events if not changed or being changed. default: null
mediaInRollback/storage/storage/unit1/cash/capabilities/types/cashOutRetract Retract unit. Items can be retracted into this unit during Cash Out operations. May be null in command data or events if not changed or being changed. default: null
mediaInRollback/storage/storage/unit1/cash/capabilities/types/reject Reject unit. Items can be rejected into this unit. May be null in command data or events if not changed or being changed. default: null
mediaInRollback/storage/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. May be null in command data if not being changed. May be null in command data or events if not changed or being changed. default: null
mediaInRollback/storage/storage/unit1/cash/capabilities/items/fit The storage unit can store cash items which are fit for recycling. May be null in command data or events if not changed or being changed. default: null
mediaInRollback/storage/storage/unit1/cash/capabilities/items/unfit The storage unit can store cash items which are unfit for recycling. May be null in command data or events if not changed or being changed. default: false
mediaInRollback/storage/storage/unit1/cash/capabilities/items/unrecognized The storage unit can store unrecognized cash items. May be null in command data or events if not changed or being changed. default: null
mediaInRollback/storage/storage/unit1/cash/capabilities/items/counterfeit The storage unit can store counterfeit cash items. May be null in command data or events if not changed or being changed. default: null
mediaInRollback/storage/storage/unit1/cash/capabilities/items/suspect The storage unit can store suspect counterfeit cash items. May be null in command data or events if not changed or being changed. default: null
mediaInRollback/storage/storage/unit1/cash/capabilities/items/inked The storage unit can store cash items which have been identified as ink stained. May be null in command data or events if not changed or being changed. default: null
mediaInRollback/storage/storage/unit1/cash/capabilities/items/coupon Storage unit containing coupons or advertising material. May be null in command data or events if not changed or being changed. default: null
mediaInRollback/storage/storage/unit1/cash/capabilities/items/document Storage unit containing documents. May be null in command data or events if not changed or being changed. default: null

Properties
mediaInRollback/storage/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. May be null in command data or events if not changed or being changed. default: null
mediaInRollback/storage/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. May be null if items can not be retracted into this storage unit or in events if not changed or being changed. Property value constraints: `minimum: 1` default: null
mediaInRollback/storage/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. May be null if items can not be retracted into this storage unit or in events if not changed or being changed. default: null
mediaInRollback/storage/storage/unit1/cash/capabilities/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. May be null in command data or events if not being modified. Property value constraints: `minItems: 1` default: null
mediaInRollback/storage/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. May be null in command data or events if no configuration is to be or has been changed. default: null
mediaInRollback/storage/storage/unit1/cash/configuration/currency ISO 4217 currency identifier [Ref. cashmanagement-1]. May only be modified in an exchange state if applicable. May be null if the unit is configured to store mixed currencies or non-cash items. Property value constraints: `pattern: ^[A-Z]{3}$` default: null
mediaInRollback/storage/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. May be null in command data or events if not being modified. Property value constraints: `minimum: 0` default: null

Properties
mediaInRollback/storage/storage/unit1/cash/configuration/highThreshold If specified, replenishmentStatus is 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 null, high is based on hardware sensors if supported - see hardwareSensors. May be null in command data or events if not being modified. Property value constraints: `minimum: 1` default: null
mediaInRollback/storage/storage/unit1/cash/configuration/lowThreshold If specified, replenishmentStatus is 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 null, low is based on hardware sensors if supported - see hardwareSensors. May be null in command data or events if not being modified. Property value constraints: `minimum: 1` default: null
mediaInRollback/storage/storage/unit1/cash/configuration/appLockIn If true, items cannot be accepted into the storage unit in Cash In operations. May be null in command data or events if not being modified. default: null
mediaInRollback/storage/storage/unit1/cash/configuration/appLockOut If true, items cannot be dispensed from the storage unit in Cash Out operations. May be null in command data or events if not being modified. default: null
mediaInRollback/storage/storage/unit1/cash/configuration/name Application configured name of the unit. May be null in command data or events if not being modified. default: null
mediaInRollback/storage/storage/unit1/cash/configuration/maxRetracts If specified, this is the number of retract operations allowed into the unit. Only applies to retract units. If retractOperations equals this number, then no further retracts are allowed into this storage unit. If null in output, the maximum number is not limited by counts. May be null in command data or events if not being modified. Property value constraints: `minimum: 1` default: null
mediaInRollback/storage/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. May be null in events if not changing. default: null
mediaInRollback/storage/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`
mediaInRollback/storage/storage/unit1/cash/status/initial The cash related items which were in the storage unit at the last replenishment. default: null

Properties
mediaInRollback/storage/storage/unit1/cash/status/initial/unrecognized Count of unrecognized items handled by the cash interface. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
mediaInRollback/storage/storage/unit1/cash/status/initial/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification. default: null
mediaInRollback/storage/storage/unit1/cash/status/initial/type20USD1/fit Count of genuine cash items which are fit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
mediaInRollback/storage/storage/unit1/cash/status/initial/type20USD1/unfit Count of genuine cash items which are unfit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
mediaInRollback/storage/storage/unit1/cash/status/initial/type20USD1/suspect Count of suspected counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
mediaInRollback/storage/storage/unit1/cash/status/initial/type20USD1/counterfeit Count of counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
mediaInRollback/storage/storage/unit1/cash/status/initial/type20USD1/inked Count of cash items which have been identified as ink stained. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null

Properties
mediaInRollback/storage/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. May be null if items have not or can not be moved from the storage unit by cash commands. default: null
mediaInRollback/storage/storage/unit1/cash/status/out/presented The items dispensed from this storage unit which are or were customer accessible. Will be null if no items were presented. default: null
mediaInRollback/storage/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. Will be null if no items were rejected. default: null
mediaInRollback/storage/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. Will be null if no items were distributed. default: null
mediaInRollback/storage/storage/unit1/cash/status/out/unknown The items dispensed from this storage unit which moved to an unknown position. Will be null if no items were unknown. default: null
mediaInRollback/storage/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. Will be null if no items were stacked. default: null
mediaInRollback/storage/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. Will be null if no items were diverted. default: null
mediaInRollback/storage/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. Will be null if no items apply. default: null

Properties
mediaInRollback/storage/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. May be null if items have not or can not be moved into the storage unit by cash commands. default: null
mediaInRollback/storage/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. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
mediaInRollback/storage/storage/unit1/cash/status/in/deposited The items deposited in the storage unit during a Cash In transaction. Can be null, if all values are 0. default: null
mediaInRollback/storage/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*. Can be null, if all values are 0. default: null
mediaInRollback/storage/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. Can be null, if all values are 0. default: null
mediaInRollback/storage/storage/unit1/cash/status/in/distributed The items deposited in this storage unit originating from another storage unit but not rejected. Can be null, if all values are 0. default: null
mediaInRollback/storage/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. Can be null, if all values are 0. default: null

Properties
mediaInRollback/storage/storage/unit1/cash/status/accuracy Describes the accuracy of the counts reported by *out* and *in*. If null in Storage.GetStorage, the hardware is not capable of determining the accuracy, otherwise the following values are possible: • `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. default: null
mediaInRollback/storage/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. May be null in events if not changing, otherwise 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. default: null
mediaInRollback/storage/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 null in Storage.GetStorage, *status* and *replenishmentStatus* apply to both cash out and cash in operations. default: null
mediaInRollback/storage/storage/unit1/card The card related contents, status and configuration of the unit. May be null if not applicable. default: null
mediaInRollback/storage/storage/unit1/card/capabilities Indicates the card storage unit capabilities. This property can be null if a change is being reported using Storage.StorageChangedEvent or Storage.StorageThresholdEvent.

Properties
mediaInRollback/storage/storage/unit1/card/capabilities/type The type of card storage. This property may be null in events if the type did not change, otherwise will be one of the following values: • `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. default: null
mediaInRollback/storage/storage/unit1/card/capabilities/hardwareSensors Indicates whether the storage unit has hardware sensors that can detect threshold states. This property may be null in events if it did not change. default: null
mediaInRollback/storage/storage/unit1/card/configuration Indicates the card storage unit configuration. This property can be null if the storage unit is being set using Storage.SetStorage, or a change is being reported using Storage.StorageChangedEvent or Storage.StorageThresholdEvent.
mediaInRollback/storage/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 storage units and may be null in events if it did not change. default: null
mediaInRollback/storage/storage/unit1/card/configuration/threshold If the threshold value is non zero, hardware sensors in the storage unit do not trigger Storage.StorageThresholdEvent events. This property may be null in events if it did not change. 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` default: null
mediaInRollback/storage/storage/unit1/card/status Indicates the card storage unit status. This property can be null if a change is being reported using Storage.StorageChangedEvent or Storage.StorageThresholdEvent.
mediaInRollback/storage/storage/unit1/card/status/initialCount The initial number of cards in the storage unit. This is only applicable to dispense type storage units. This property may be null in events if it did not change. This value is persistent. Property value constraints: `minimum: 0` default: null

Properties
mediaInRollback/storage/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 and may be null in events if it did not change. Property value constraints: `minimum: 0` default: null
mediaInRollback/storage/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 and may be null in events if it did not change. Property value constraints: `minimum: 0` default: null
mediaInRollback/storage/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 null. The property may also be null in events if it did not change. 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. default: null
mediaInRollback/storage/storage/unit1/check The check related contents, status and configuration of the unit. May be null if not applicable. default: null
mediaInRollback/storage/storage/unit1/check/capabilities Indicates what the storage unit is capable of - this includes information which is a combination of that reported in WFS_INF_IPM_MEDIA_BIN_INFO and WFS_INF_IPM_MEDIA_BIN_CAPABILITIES in XFS 3.x. May be null in events if not changed. default: null
mediaInRollback/storage/storage/unit1/check/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. May be null in command data and events if not changing. default: null

Properties
mediaInRollback/storage/storage/unit1/check/capabilities/types/mediaIn The unit can accept items during Media In transactions. May be null in command data and events if not changing. default: null
mediaInRollback/storage/storage/unit1/check/capabilities/types/retract Retract unit. Items can be retracted into this unit using Check.RetractMedia. May be null in command data and events if not changing. default: null
mediaInRollback/storage/storage/unit1/check/capabilities/sensors The types of sensor the unit has. May be null in command data and events if not changing. default: null
mediaInRollback/storage/storage/unit1/check/capabilities/sensors/empty The unit contains a hardware sensor which reports when the unit is empty. May be null in command data and events if not changing. default: null
mediaInRollback/storage/storage/unit1/check/capabilities/sensors/high The unit contains a hardware sensor which reports when the unit is nearly full. May be null in command data and events if not changing. default: null
mediaInRollback/storage/storage/unit1/check/capabilities/sensors/full The unit contains a hardware sensor which reports when the unit is full. May be null in command data and events if not changing. default: null
mediaInRollback/storage/storage/unit1/check/configuration Indicates what the storage unit is configured to do - where applicable the supported options can be derived from capabilities. May be null in command data and events if not being modified.
mediaInRollback/storage/storage/unit1/check/configuration/binID An application defined Storage Unit Identifier. This may be null in events if not changing. default: null
mediaInRollback/storage/storage/unit1/check/configuration/highThreshold If specified, replenishmentStatus is set to *high* if the total number of items in the storage unit is greater than this number. May be null in command data and events if not being modified. Property value constraints: `minimum: 1` default: null
mediaInRollback/storage/storage/unit1/check/configuration/retractHighThreshold If specified and the storage unit is configured as *retract*, replenishmentStatus is set to *high* if the total number of retract operations in the storage unit is greater than this number. May be null in command data and events if not being modified. Property value constraints: `minimum: 0` default: null
mediaInRollback/storage/storage/unit1/check/status Indicates the storage unit status. May be null in events where status has not changed. default: null

Properties
mediaInRollback/storage/storage/unit1/check/status/index Assigned by the Service. Will be a unique number which can be used to determine *usBinNumber* in XFS 3.x migration. This can change as storage units are added and removed from the storage collection. Property value constraints: `minimum: 1`
mediaInRollback/storage/storage/unit1/check/status/initial The check related counts as set at the last replenishment. May be null in events where status has not changed. default: null
mediaInRollback/storage/storage/unit1/check/status/initial/mediaInCount Count of items added to the storage unit due to Check operations. If the number of items is not counted this is not reported and *retractOperations* is incremented as items are added to the unit. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
mediaInRollback/storage/storage/unit1/check/status/initial/count Total number of items added to the storage unit due to any operations. If the number of items is not counted this is not reported and *retractOperations* is incremented as items are added to the unit. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
mediaInRollback/storage/storage/unit1/check/status/initial/retractOperations Total number of operations which resulted in items being retracted to the storage unit. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
mediaInRollback/storage/storage/unit1/check/status/in The check items added to the unit since the last replenishment. May be null in events where status has not changed. default: null
mediaInRollback/storage/storage/unit1/check/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. May be null in command data and events if not changing, otherwise 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 a full sensor or counts. • `high` - The storage unit is almost full (either high sensor based or exceeded the highThreshold or retractHighThreshold). • `empty` - The storage unit is empty, or insufficient items in the storage unit are preventing further dispense operations. If the storage unit has the empty sensor, this state is not set by counts. default: null

Event Messages

- Check.MediaPresentedEvent

9.2.5 Check.ReadImage

On devices where items can be physically rescanned or all the supported image formats can be generated during this command (regardless of the images requested during the Check.MediaIn command), i.e. where rescan is true, then this command is used to obtain additional images and/or reread the code line for media already in the device.

If *rescan* is false, this command is used to retrieve an image or code line that was initially obtained when the media was initially processed (e.g. during the *Check.MediaIn* or Check.GetNextItem command). In this case, all images required must have been previously been requested during the *Check.MediaIn* command.

The media has to be inserted using the command *Check.MediaIn*. If no media is present the command returns the error code *noMediaPresent*.

Command Message

Payload (version 2.0)	Type	Required
{		
"mediaID": 4,	integer	✓
"codelineFormat": "e13b",	string, null	
"image": [{	array (object), null	
"source": "back",	string	✓
"type": "jpg",	string	✓
"colorFormat": "full",	string	✓
"scanColor": "white"	string	✓
}]		
}		

Properties

mediaID

Specifies the sequence number (starting from 1) of a media item.

Property value constraints:

```
minimum: 1
```

codelineFormat

Specifies the code line format. May be null if no code line data is required. If supplied, it must be one of the supported code line formats. The following values are possible:

- `cmc7` - Read CMC7 code line [Ref. check-4].
- `e13b` - Read E13B code line [Ref. check-3].
- `ocr` - Read code line using OCR. The default or pre-configured OCR font will be used.
- `ocra` - Read code line using OCR font A [Ref. check-1].
- `ocrb` - Read code line using OCR font B [Ref. check-2].

default: null

image

An array specifying the images to be read for each item. May be null if no images are required.

default: null

image/source

Specifies the source. The following values are possible:

- `front` - The image is for the front of the media item.
- `back` - The image is for the back of the media item.

Properties
image/type Specifies the format of the image. The following values are possible: • `tif` - The image is in TIFF 6.0 format. • `wmf` - The image is in WMF (Windows Metafile) format. • `bmp` - The image is in Windows BMP format. • `jpg` - The image is in JPG format.
image/colorFormat Specifies the color format of the image. The following values are possible: • `binary` - The image is binary (image contains two colors, usually the colors black and white). • `grayScale` - The image is gray scale (image contains multiple gray colors). • `full` - The image is full color (image contains colors like red, green, blue etc.).
image/scanColor Selects the scan color. The following values are possible: • `red` - The image is scanned with red light. • `green` - The image is scanned with green light. • `blue` - The image is scanned with blue light. • `yellow` - The image is scanned with yellow light. • `white` - The image is scanned with white light. • `infraRed` - The image is scanned with infrared light. • `ultraViolet` - The image is scanned with ultraviolet light.

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "invalidMediaID",	string, null	
"data": {	object, null	
"mediaID": 4,	integer	✓
"codelineData": "□22222□□123456□",	string	
"magneticReadIndicator": "noMicr",	string	✓
"image": [{	array (object), null	
"imageSource": "back",	string	✓
"imageType": "jpg",	string	✓
"imageColorFormat": "full",	string	✓
"imageScanColor": "white",	string	✓
"imageStatus": "ok",	string	✓
"image": "wCAAAQgwMDAwMDAwMA=="	string, null	
}],		
"insertOrientation": {	object, null	
"codeline": "top",	string, null	
"media": "down"	string, null	
},		
"mediaSize": {	object, null	
"longEdge": 205,	integer	

Payload (version 2.0)	Type	Required
"shortEdge": 103	integer	
},		
"mediaValidity": "ok"	string	
}		
}		

Properties

errorCode

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

- `mediaJammed` - The media is jammed.
- `scannerInop` - Only images were requested by the application and these cannot be obtained because the image scanner is inoperative.
- `micrInop` - Only MICR data was requested by the application and it cannot be obtained because the MICR reader is inoperative.
- `noMedia` - No media is present in the device.
- `invalidMediaID` - The requested media ID does not exist.

default: null

data

Image data. May be null if an error occurred.

default: null

data/mediaID

Specifies the sequence number (starting from 1) of a media item.

Property value constraints:

```
minimum: 1
```

data/codelineData

Specifies the code line data. See Code line Characters.

default: ""

data/magneticReadIndicator

Specifies the type of technology used to read a MICR code line. The following values are possible:

- `micr` - The MICR code line was read using MICR technology and MICR characters were present.
- `notMicr` - The MICR code line was NOT read using MICR technology.
- `noMicr` - The MICR code line was read using MICR technology and no magnetic characters were read.
- `unknown` - It is unknown how the MICR code line was read.
- `notMicrFormat` - The code line is not a MICR format code line.
- `notRead` - No code line was read.

data/image

Array of image data. If the Device has determined the orientation of the media (i.e. *insertOrientation* is defined and not set to "unknown"), then all images returned are in the standard orientation and the images will match the image source requested by the application. This means that images will be returned with the code line at the bottom, and the image of the front and rear of the media item will be returned in the structures associated with the "front" and "back" image sources respectively.

default: null

data/image/imageSource

Specifies the source. The following values are possible:

- `front` - The image is for the front of the media item.
- `back` - The image is for the back of the media item.

Properties

data/image/imageType

Specifies the format of the image. The following values are possible:

- `tif` - The image is in TIFF 6.0 format.
- `wmf` - The image is in WMF (Windows Metafile) format.
- `bmp` - The image is in Windows BMP format.
- `jpg` - The image is in JPG format.

data/image/imageColorFormat

Specifies the color format of the image. The following values are possible:

- `binary` - The image is binary (image contains two colors, usually the colors black and white).
- `grayScale` - The image is gray scale (image contains multiple gray colors).
- `full` - The image is full color (image contains colors like red, green, blue etc.).

data/image/imageScanColor

Selects the scan color. The following values are possible:

- `red` - The image is scanned with red light.
- `green` - The image is scanned with green light.
- `blue` - The image is scanned with blue light.
- `yellow` - The image is scanned with yellow light.
- `white` - The image is scanned with white light.
- `infraRed` - The image is scanned with infrared light.
- `ultraViolet` - The image is scanned with ultraviolet light.

data/image/imageStatus

Status of the image data. The following values are possible:

- `ok` - The data is OK.
- `sourceNotSupported` - The data source or image attributes are not supported by the Service, e.g., scan color not supported.
- `sourceMissing` - The image could not be obtained.

data/image/image

Base64 encoded image. May be null if no image was obtained.

Property value constraints:

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

default: null

data/insertOrientation

This value reports how the media item was actually inserted into the input position (from the customer's perspective). The full orientation can be determined as a combination of *codeline* and *media* values. If the orientation is unknown, this will be null.

default: null

data/insertOrientation/codeline

Specifies the orientation of the code line. The following values are possible, or null if unknown.

- `right` - The code line is to the right.
- `left` - The code line is to the left.
- `bottom` - The code line is to the bottom.
- `top` - The code line is to the top.

default: null

data/insertOrientation/media

Specifies the orientation of the media. The following values are possible, or null if unknown:

- `up` - The front of the media (the side with the code line) is facing up.
- `down` - The front of the media (the side with the code line) is facing down.

default: null

Properties
data/mediaSize Specifies the size of the media item. Will be null if the device does not support media size measurement or no size measurements are known. default: null
data/mediaSize/longEdge Specifies the length of the long edge of the media in millimeters, or 0 if unknown. Property value constraints: `minimum: 0` default: 0
data/mediaSize/shortEdge Specifies the length of the short edge of the media in millimeters, or 0 if unknown. Property value constraints: `minimum: 0` default: 0
data/mediaValidity Media items may have special security features which can be detected by the device. This specifies whether the media item is suspect or valid, allowing the application the choice in how to further process a media item that could not be confirmed as being valid. The following values are possible: • `ok` - The media item is valid. • `suspect` - The validity of the media item is suspect. • `unknown` - The validity of the media item is unknown. • `noValidation` - No specific security features were evaluated. default: "ok"

Event Messages

None

9.2.6 Check.PresentMedia

This command is used to present media items to the customer.

Applications can use this command to return refused items without terminating the media-in transaction. This allows customers to correct the problem with the media item and reinsert during execution of a subsequent Check.MediaIn command.

This command is also used to return items after a Check.MediaInEnd or Check.MediaInRollBack command when presentControl is false.

A Check.MediaPresentedEvent event is generated when media is presented and a Check.MediaTakenEvent event is generated when the media is taken (if the position has a taken sensor itemsTakenSensor is true.

This command completes when the last bunch of media items to be returned to the customer has been presented, but before the last bunch is taken.

Command Message

Payload (version 2.0)	Type	Required
{		
"source": {	object, null	
"position": "refused"	string	✓
}		
}		

Properties

source
Specifies the position where items are to be returned from. If null, all items are returned from all positions in a sequence determined by the Service, otherwise an individual *position* must be specified.
default: null

source/position
Specifies the position.
It is specified as one of the following values:

- `input` - The input position.
- `refused` - The refused media position.
- `rebuncher` - The refuse/return re-buncher.

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "mediaJammed"	string, null	
}		

Properties

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

- `noMedia` - No media is present in the device.
- `shutterFail` - Open or close of the shutter failed due to manipulation or hardware error.
- `mediaJammed` - The media is jammed.
- `positionNotEmpty` - One of the input/output/refused positions is not empty and items cannot be inserted until the media items in the position are removed.

default: null

Event Messages

- Check.MediaPresentedEvent

9.2.7 Check.RetractMedia

The media is removed from its present position (media present in device, media entering, unknown position) and stored in the area specified in the input parameters.

A Storage.StorageThresholdEvent event is sent if a high or full condition is reached as a result of this command. If the storage unit is already full and the command cannot be executed, an error is returned and the media remains in its present position.

If media items are to be endorsed/stamped during this operation, then the SetMediaParameters command must be called prior to the Check.RetractMedia command. Where endorsing is specified, the same text will be printed on all media items that are detected.

This command ends the current media-in transaction.

If no items are in the device the command will complete with the *noMediaPresent* error and the mediaInTransaction will be set to *retract*.

Command Message

Payload (version 2.0)	Type	Required
{		
"retractLocation": "rebuncher"	string	✓
}		

Properties

retractLocation

Specifies the location for the retracted media, on input where it is to be retracted to, on output where it was retracted to. See retractLocation to determine the supported locations. This can take one of the following values:

- `stacker` - The device stacker.
- `transport` - The device transport.
- `rebuncher` - The device rebuncher.
- `<storage unit identifier>` - A storage unit as specified by identifier.

Property value constraints:

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

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "shutterFail",	string, null	
"media": {	object, null	
"count": "1",	string	
"retractLocation": "rebuncher"	string	✓
}		
}		

Properties
errorCode Specifies the error code if applicable, otherwise null. Following values are possible: • `noMediaPresent` - No media present on retract. Either there was no media present (in a position to be retracted) when the command was called or the media was removed during the retract. • `mediaJammed` - The media is jammed. Operator intervention is required. • `stackerFull` - The stacker or re-buncher is full. • `invalidBin` - The specified storage unit cannot accept retracted items. • `noBin` - The specified storage unit does not exist. • `mediaBinError` - A storage unit caused a problem. A Storage.StorageErrorEvent will be posted with the details. • `shutterFail` - Open or close of the shutter failed due to manipulation or hardware error. • `foreignItemsDetected` - Foreign items have been detected in the input position. default: null
media The details of the media retracted. May be null if no media was retracted. default: null
media/count Contains the number of media items retracted as a result of this command. The following values are possible: • `<number>` - The number of items retracted. • `unknown` - The number of items is unknown. Property value constraints: `pattern: ^unknown$\|^[0-9]+$`
media/retractLocation Specifies the location for the retracted media, on input where it is to be retracted to, on output where it was retracted to. See retractLocation to determine the supported locations. This can take one of the following values: • `stacker` - The device stacker. • `transport` - The device transport. • `rebuncher` - The device rebuncher. • `<storage unit identifier>` - A storage unit as specified by identifier. Property value constraints: `pattern: ^stacker$\|^transport$|^rebuncher$\|^unit[0-9A-Za-z]+$`

Event Messages

- Storage.StorageErrorEvent

9.2.8 Check.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 service handle.

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. One or more Check.MediaDetectedEvent events will inform the application where items were actually moved to.

If media items are to be endorsed/stamped during this operation, then the SetMediaParameters must be called prior to the Check.Reset command. Where endorsing is specified, the same text will be printed on all media items that are detected.

This command ends a media-in transaction started by the Check.MediaIn command.

Command Message

Payload (version 2.0)	Type	Required
{		
"mediaControl": "transport"	string	✓
}		

Properties

mediaControl

Specifies the manner in which the media should be handled, as one of the following values. See resetControl to determine the supported options.

- `eject` - Eject the media, i.e. return the media to the customer. Note that more than one position may be used to return media.
- `<storage unit identifier>` - The media item is in a storage unit as specified by identifier.
- `transport` - Retract the media to the transport.
- `rebuncher` - Retract the media to the rebuncher.

Property value constraints:

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

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "shutterFail"	string, null	
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.
- `mediaBinError` - A storage unit caused a problem. A Storage.StorageErrorEvent will be posted with the details.
- `invalidBin` - The specified storage unit cannot accept retracted items.

default: null

Event Messages

- Storage.StorageErrorEvent
- Check.MediaPresentedEvent

9.2.9 Check.GetNextItem

This command is used to get the next item from the multi-item feed unit and capture the item data. The data and the format of the data that is generated by this command are defined by the input parameters of the Check.MediaIn command. The media data is reported via the Check.MediaDataEvent event.

This command must be supported by all Services where the hardware does not have a stacker or where the Service supports the application making the accept/refuse decision. On single item feed devices this command simply returns the error code *noMediaPresent*. This allows a single application flow to be used on all devices without a stacker.

Command Message

Payload (version 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "noItems",`	string, null	
`"mediaFeeder": "notEmpty"`	string, null	
`}`		

Properties

errorCode

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

- `noItems` - There were no items present in the device.
- `mediaJammed` - The media is jammed.
- `refusedItems` - Programming error, refused items that must be returned have not been presented.
- `positionNotEmpty` - One of the input/output/refused positions is not empty.
- `scannerInop` - Only images were requested by the application and these cannot be obtained because the image scanner is inoperative.
- `micrInop` - Only MICR data was requested by the application and it cannot be obtained because the MICR reader is inoperative.
- `feederInop` - The media feeder is inoperative.

default: null

mediaFeeder

Supplies the state of the media feeder. This value indicates if there are items on the media feeder waiting for processing via the Check.GetNextItem command. If null, the device has no media feeder or the capability to report the status of the media feeder is not supported by the device. This value can be one of the following values:

- `empty` - The media feeder is empty.
- `notEmpty` - The media feeder is not empty.
- `inoperative` - The media feeder is inoperative.
- `unknown` - Due to a hardware error or other condition, the state of the media feeder cannot be determined.

default: null

Event Messages

- Check.MediaRefusedEvent
- Check.MediaDataEvent

9.2.10 Check.ActionItem

This command is used to cause the predefined actions (move item to destination, stamping, endorsing, re-imaging) to be executed on the current media item. This command only applies to devices without stackers and on devices with stackers this command is not supported.

Command Message

Payload (version 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "inkOut"`	string, null	
`}`		

Properties

errorCode

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

- `shutterFail` - Open or close of the shutter failed due to manipulation or hardware error.
- `mediaBinError` - A storage unit caused a problem. A Storage.StorageErrorEvent

will be posted with the details.

- `mediaJammed` - The media is jammed.
- `tonerOut` - Toner or ink supply is empty or printing contrast with ribbon is not sufficient.
- `inkOut` - No stamping possible, stamping ink supply empty.
- `noMediaPresent` - There were no items present in the device.
- `scannerInop` - The scanner is inoperative.
- `refusedItems` - Programming error, refused items that must be returned have not been presented.
- `positionNotEmpty` - One of the input/output/refused positions is not empty.

default: null

Event Messages

- Check.MediaPresentedEvent
- Check.MediaDataEvent
- Storage.StorageErrorEvent

9.2.11 Check.ExpelMedia

The media that has been presented to the customer will be expelled out of the device.

This command completes after the bunch has been expelled from the device.

This command does not end the current media-in transaction. The application must deal with any media remaining within the device, e.g., by using the Check.MediaInRollBack, Check.MediaInEnd, or Check.RetractMedia command.

Command Message

Payload (version 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "shutterFail"`	string, null	
`}`		

Properties

errorCode

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

- `shutterFail` - Open or close of the shutter failed due to manipulation or hardware error.
- `mediaJammed` - The media is jammed.
- `noItems` - There were no items present in the device to expel.

default: null

Event Messages

None

9.2.12 Check.AcceptItem

This command is used by applications to indicate if the current media item should be accepted or refused. Applications only use this command when the Check.MediaIn command is used in the mode where the application can decide if each physically acceptable media item should be accepted or refused, i.e. applicationRefuse is true.

Command Message

Payload (version 2.0)	Type	Required
{		
"accept": true	boolean	✓
}		

Properties

accept
Specifies if the item should be accepted or refused. If true then the item is accepted and moved to the stacker. If false then the item is moved to the re-buncher/refuse position.

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "mediaJammed"	string, null	
}		

Properties

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

- `mediaJammed` - The media is jammed.
- `noItems` - There were no items present in the device.
- `refusedItems` - Programming error, refused items that must be returned have not been presented.
- `positionNotEmpty` - One of the input/output/refused positions is not empty.

default: null

Event Messages

None

9.2.13 Check.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. A Common.StatusChangedEvent event must be broadcast to report that the state has changed.

Hardware that can detect the level of a supply must update its status based on its sensors, generate a Common.StatusChangedEvent 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 new status before this command was issued, the command must succeed and no event is required.

Command Message

Payload (version 2.0)	Type	Required
{		
"toner": true,	boolean	
"ink": false	boolean	
}		

Properties

toner
Specifies whether the toner supply was replenished.
default: false

ink
Specifies whether the ink supply was replenished.
default: false

Completion Message

Payload (version 2.0)
This message does not define any properties.

Event Messages

None

9.2.14 Check.SetMediaParameters

This command is used to predefine parameters for the specified media item or all items. The command can be called multiple times to specify individual parameters for each required media item. Any parameter specified replaces any parameters specified for the same media item (or items) on previous commands.

The parameters which can be specified include:

- Destination
- Endorsements, i.e., text to be printed on the media or whether the media is to be stamped
- Images of the media after it has been printed on or stamped

The media is not moved immediately by this command. The requested actions are performed during subsequent commands which move the media:

- On devices with stackers, Check.MediaInEnd
- On devices without stackers, Check.ActionItem

If the bunch is returned with Check.MediaInRollback, none of the requested actions will be performed.

If the media is to be returned to the customer using *Check.MediaInEnd* or *Check.ActionItem*, the media can still be endorsed if endorse is true or imaged if endorseImage is true.

The Service will determine which storage unit to use for any items that have not had a destination set by the application.

Command Message

Payload (version 2.0)	Type	Required
`{`		
`"mediaID": 4,`	integer, null	
`"destination": "unit1",`	string	✓
`"stamp": false,`	boolean	
`"printData": "Text to print on the check.",`	string	
`"image": [{`	array (object), null	
`"source": "back",`	string	✓
`"type": "jpg",`	string	✓
`"colorFormat": "full",`	string	✓
`"scanColor": "white"`	string	✓
`}]`		
`}`		

Properties

mediaID

Specifies the sequence number of a media item. Valid IDs are 1 to the maximum media ID assigned within the transaction - see mediaID. If null, all media items are selected.

Property value constraints:

```
minimum: 1
```

default: null

Properties
destination Specifies where the item(s) specified by *mediaID* are to be moved to. Following values are possible: • `customer` - The item or items are to be returned to the customer. • `<storage unit identifier>` - The item or items are to be sored in a storage unit with matching identifier. Property value constraints: `pattern: ^customer$\|^unit[0-9A-Za-z]+$`
stamp Specifies whether the media will be stamped. If not specified, the media will not be stamped. default: false
printData Specifies the data that will be printed on the media item that is entered by the customer. If a character is not supported by the device it will be replaced by a vendor dependent substitution character. If not specified, no text will be printed. For devices that can print multiple lines, each line is separated by a Carriage Return (Unicode 0x000D) and Line Feed (Unicode 0x000A) sequence. For devices that can print on both sides, the front and back print data are separated by a Carriage Return (Unicode 0x000D) and a Form Feed (Unicode 0x000C) sequence. In this case the data to be printed on the back is the first set of data, and the front is the second set of data. default: ""
image Specifies the images required. May be null if no images are required. default: null
image/source Specifies the source. The following values are possible: • `front` - The image is for the front of the media item. • `back` - The image is for the back of the media item.
image/type Specifies the format of the image. The following values are possible: • `tif` - The image is in TIFF 6.0 format. • `wmf` - The image is in WMF (Windows Metafile) format. • `bmp` - The image is in Windows BMP format. • `jpg` - The image is in JPG format.
image/colorFormat Specifies the color format of the image. The following values are possible: • `binary` - The image is binary (image contains two colors, usually the colors black and white). • `grayScale` - The image is gray scale (image contains multiple gray colors). • `full` - The image is full color (image contains colors like red, green, blue etc.).
image/scanColor Selects the scan color. The following values are possible: • `red` - The image is scanned with red light. • `green` - The image is scanned with green light. • `blue` - The image is scanned with blue light. • `yellow` - The image is scanned with yellow light. • `white` - The image is scanned with white light. • `infraRed` - The image is scanned with infrared light. • `ultraViolet` - The image is scanned with ultraviolet light.

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "invalidMediaID"	string, null	
}		

Properties

errorCode

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

- `invalidMediaID` - The requested media ID does not exist.
- `invalidBin` - The specified storage unit cannot accept items.
- `noBin` - The specified storage unit does not exist.
- `mediaBinFull` - The storage unit is already full and no media can be placed in the specified storage unit.
- `mediaJammed` - The media is jammed.
- `scannerInop` - Only images were requested by the application and these cannot be obtained because the image scanner is inoperative.
- `noItems` - There were no items present in the device.
- `tonerOut` - Toner or ink supply is empty or printing contrast with ribbon is not sufficient.
- `inkOut` - No stamping possible, stamping ink supply empty.

default: null

Event Messages

None

9.3 Event Messages

9.3.1 Check.NoMediaEvent

This reports that the physical media must be inserted into the device in order for the command to proceed.

Event Message

Payload (version 2.0)
This message does not define any properties.

9.3.2 Check.MediaInsertedEvent

This specifies that the physical media has been inserted into the device.

Event Message

Payload (version 2.0)
This message does not define any properties.

9.3.3 Check.MediaRefusedEvent

This is sent when a media item is refused. One message is sent for every media item or bunch of media items that has been refused.

Event Message

Payload (version 2.0)	Type	Required
{		
"reason": "metal",	string	✓
"location": "refused",	string	✓
"presentRequired": true,	boolean	✓
"mediaSize": {	object, null	
"longEdge": 205,	integer	
"shortEdge": 103	integer	
}		
}		

Properties

reason

Specified the reason why the media was refused. Specified as one of the following values:

- `foreignItems` - Foreign items were detected in the input position.
- `stackerFull` - The stacker is full or the maximum number of items that the application wants to be allowed on the stacker has been reached (see usMaxMediaOnStacker input parameter in the WFS_CMD_IPM_MEDIA_IN command).
- `codelineInvalid` - The code line data was found but was invalid.
- `invalidMedia` - The media item is not a check, and in the case of Mixed Media processing not a cash item either.
- `tooLong` - The media item (or bunch of items) long edge was too long.
- `tooShort` - The media item (or bunch of items) long edge was too short.
- `tooWide` - The media item (or bunch of items) short edge was too wide.
- `tooNarrow` - The media item (or bunch of items) short edge was too narrow.
- `tooThick` - The media item was too thick.
- `invalidOrientation` - The media item was inserted in an invalid orientation.
- `doubleDetect` - The media items could not be separated.
- `refusePosFull` - There are too many items in the refuse area. The refused items must be returned to the customer before any additional media items can be accepted.
- `returnBlocked` - Processing of the items did not take place as the bunch of items is blocking the return of other items.
- `invalidBunch` - Processing of the items did not take place as the bunch of items presented is invalid, e.g. it is too large or was presented incorrectly.
- `otherItem` - The item was refused for some reason not covered by the other reasons.
- `otherBunch` - The bunch was refused for some reason not covered by the other reasons.
- `jamming` - The media item is causing a jam.
- `metal` - Metal (e.g. staple, paperclip, etc) was detected in the input position.

location

Specifies where the refused media should be presented to the customer from. It can be one of the following values:

- `input` - The input position.
- `refused` - The refused media position.
- `rebuncher` - The refuse/return re-buncher.
- `stacker` - The stacker.

Properties
presentRequired This indicates whether the media needs to be presented to the customer before any additional media movement commands can be executed. If true then the media must be presented to the customer via the Check.PresentMedia command before further media movement commands can be executed. If false then the device can continue without the media being returned to the customer.
mediaSize Specifies the size of the media item. Will be null if the device does not support media size measurement or no size measurements are known. default: null
mediaSize/longEdge Specifies the length of the long edge of the media in millimeters, or 0 if unknown. Property value constraints: `minimum: 0` default: 0
mediaSize/shortEdge Specifies the length of the short edge of the media in millimeters, or 0 if unknown. Property value constraints: `minimum: 0` default: 0

9.3.4 Check.MediaDataEvent

This message returns the code line and all the images requested for each item processed. This can be generated during the Check.MediaIn, Check.MediaInEnd, Check.GetNextItem and Check.ActionItem commands. One message is generated for each item processed, no message is generated for refused items.

Event Message

Payload (version 2.0)	Type	Required
{		
"mediaID": 4,	integer	✓
"codelineData": "□22222□□123456□",	string	
"magneticReadIndicator": "noMicr",	string	✓
"image": [{	array (object), null	
"imageSource": "back",	string	✓
"imageType": "jpg",	string	✓
"imageColorFormat": "full",	string	✓
"imageScanColor": "white",	string	✓
"imageStatus": "ok",	string	✓
"image": "wCAAAQgwMDAwMDAwMA=="	string, null	
}],		
"insertOrientation": {	object, null	
"codeline": "top",	string, null	
"media": "down"	string, null	
},		
"mediaSize": {	object, null	
"longEdge": 205,	integer	
"shortEdge": 103	integer	
},		
"mediaValidity": "ok"	string	
}		

Properties

mediaID
Specifies the sequence number (starting from 1) of a media item.
Property value constraints:
`minimum: 1`

codelineData
Specifies the code line data. See Code line Characters.
default: ""

Properties
magneticReadIndicator Specifies the type of technology used to read a MICR code line. The following values are possible: • `micr` - The MICR code line was read using MICR technology and MICR characters were present. • `notMicr` - The MICR code line was NOT read using MICR technology. • `noMicr` - The MICR code line was read using MICR technology and no magnetic characters were read. • `unknown` - It is unknown how the MICR code line was read. • `notMicrFormat` - The code line is not a MICR format code line. • `notRead` - No code line was read.
image Array of image data. If the Device has determined the orientation of the media (i.e. *insertOrientation* is defined and not set to "unknown"), then all images returned are in the standard orientation and the images will match the image source requested by the application. This means that images will be returned with the code line at the bottom, and the image of the front and rear of the media item will be returned in the structures associated with the "front" and "back" image sources respectively. default: null
image/imageSource Specifies the source. The following values are possible: • `front` - The image is for the front of the media item. • `back` - The image is for the back of the media item.
image/imageType Specifies the format of the image. The following values are possible: • `tif` - The image is in TIFF 6.0 format. • `wmf` - The image is in WMF (Windows Metafile) format. • `bmp` - The image is in Windows BMP format. • `jpg` - The image is in JPG format.
image/imageColorFormat Specifies the color format of the image. The following values are possible: • `binary` - The image is binary (image contains two colors, usually the colors black and white). • `grayScale` - The image is gray scale (image contains multiple gray colors). • `full` - The image is full color (image contains colors like red, green, blue etc.).
image/imageScanColor Selects the scan color. The following values are possible: • `red` - The image is scanned with red light. • `green` - The image is scanned with green light. • `blue` - The image is scanned with blue light. • `yellow` - The image is scanned with yellow light. • `white` - The image is scanned with white light. • `infraRed` - The image is scanned with infrared light. • `ultraViolet` - The image is scanned with ultraviolet light.
image/imageStatus Status of the image data. The following values are possible: • `ok` - The data is OK. • `sourceNotSupported` - The data source or image attributes are not supported by the Service, e.g., scan color not supported. • `sourceMissing` - The image could not be obtained.

Properties

image/image
Base64 encoded image. May be null if no image was obtained.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`
default: null

insertOrientation This value reports how the media item was actually inserted into the input position (from the customer's perspective). The full orientation can be determined as a combination of *codeline* and *media* values. If the orientation is unknown, this will be null. default: null
insertOrientation/codeline Specifies the orientation of the code line. The following values are possible, or null if unknown. • `right` - The code line is to the right. • `left` - The code line is to the left. • `bottom` - The code line is to the bottom. • `top` - The code line is to the top. default: null
insertOrientation/media Specifies the orientation of the media. The following values are possible, or null if unknown: • `up` - The front of the media (the side with the code line) is facing up. • `down` - The front of the media (the side with the code line) is facing down. default: null
mediaSize Specifies the size of the media item. Will be null if the device does not support media size measurement or no size measurements are known. default: null
mediaSize/longEdge Specifies the length of the long edge of the media in millimeters, or 0 if unknown. Property value constraints: `minimum: 0` default: 0
mediaSize/shortEdge Specifies the length of the short edge of the media in millimeters, or 0 if unknown. Property value constraints: `minimum: 0` default: 0
mediaValidity Media items may have special security features which can be detected by the device. This specifies whether the media item is suspect or valid, allowing the application the choice in how to further process a media item that could not be confirmed as being valid. The following values are possible: • `ok` - The media item is valid. • `suspect` - The validity of the media item is suspect. • `unknown` - The validity of the media item is unknown. • `noValidation` - No specific security features were evaluated. default: "ok"

9.3.5 Check.MediaRejectedEvent

This reports that an attempt to insert media into the device has been rejected before the media was fully inside the device, i.e. no Check.MediaInsertedEvent event has been generated. Rejection of the media will cause the Check.MediaIn command to complete with a *mediaRejected* error, at which point the media should be removed.

Event Message

Payload (version 2.0)	Type	Required
{		
"reason": "metal"	string	✓
}		

Properties

reason

Specified the reason why the media was rejected. Specified as one of the following values:

- `long` - The media was too long.
- `thick` - The media was too thick.
- `double` - More than one media item was detected (this value only applies to devices without a media feeder).
- `transport` - The media could not be moved inside the device.
- `shutter` - The media was rejected due to the shutter failing to close.
- `removed` - The media was removed (no Check.MediaTakenEvent is expected).
- `metal` - Metal (e.g. staple, paperclip, etc) was detected in the input position.
- `foreignItems` - Foreign items were detected in the input position.
- `other` - The item was rejected for some reason not covered by the other reasons.

9.3.6 Check.MediaPresentedEvent

This indicates that media has been presented to the customer for removal.

Event Message

Payload (version 2.0)	Type	Required
{		
"position": "refused",	string	✓
"bunchIndex": 2,	integer	✓
"totalBunches": "1"	string	
}		

Properties

position

Specifies the position.

It is specified as one of the following values:

- `input` - The input position.
- `refused` - The refused media position.
- `rebuncher` - The refuse/return re-buncher.

bunchIndex

Specifies the index (starting from one) of the presented bunch (one or more items presented as a bunch).

Property value constraints:

```
minimum: 1
```

totalBunches

Specifies the total number of bunches to be returned from all positions. The total represents the number of bunches that will be returned as a result of a single command that presents media. The 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"

9.4 Unsolicited Messages

9.4.1 Check.MediaTakenEvent

This is sent when the media is taken by the customer.

Unsolicited Message

Payload (version 2.0)	Type	Required
{		
"position": "refused"	string	✓
}		

Properties

position
Specifies the position.
It is specified as one of the following values:

- `input` - The input position.
- `refused` - The refused media position.
- `rebuncher` - The refuse/return re-buncher.

9.4.2 Check.MediaDetectedEvent

This service event is generated when media is detected in the device during a Check.Reset.

Unsolicited Message

Payload (version 2.0)	Type	Required
{		
"position": "customer"	string	✓
}		

Properties

position

Specifies the media position after the operation, as one of the following values:

- `<storage unit identifier>` - The media item was retracted to a storage unit as specified by identifier.
- `device` - The media is in the device.
- `position` - The media is at one or more of the input, output and refused positions.
- `jammed` - The media is jammed in the device.
- `customer` - The media has been returned and taken by the customer.
- `unknown` - The media is in an unknown position.

Property value constraints:

```
pattern: ^device$|^position$|^jammed$|^customer$|^unknown$|^unit[0-9A-Za-z]+$
```

9.4.3 Check.ShutterStatusChangedEvent

Within the limitations of the hardware sensors this service event is generated whenever the status of a shutter changes. The shutter status can change because of an explicit, implicit or manual operation depending on how the shutter is operated.

Unsolicited Message

Payload (version 2.0)	Type	Required
{		
"position": "refused",	string	✓
"shutter": "closed"	string, null	
}		

Properties

position

Specifies the position.

It is specified as one of the following values:

- `input` - The input position.
- `refused` - The refused media position.
- `rebuncher` - The refuse/return re-buncher.

shutter

Specifies the state of the shutter. This property is null in Common.Status if the physical device has no shutter or shutter state reporting is not supported, otherwise the following values are possible:

- `closed` - The shutter is operational and is closed.
- `open` - The shutter is operational and is open.
- `jammed` - The shutter is jammed and is not operational.
- `unknown` - Due to a hardware error or other condition, the state of the shutter cannot be determined.

default: null

10. Mixed Media

This chapter provides information on how to perform MixedMedia transactions. A Mixed Media transaction is defined as a single transaction on a single device which can accept different kinds of media, specifically cash and checks.

It is possible that the device may process items in mixed bunches within one operation, or require that the different media types are processed in separate operations. The MixedMedia interface defines what type of mixed media transactions the device is capable of and configured for and methods to choose which type of transaction should be performed.

This interface would only be supported by devices which support MixedMedia functionality. A device which can accept cash and checks but not in one transaction would not support this interface.

10.1 General Information

10.1.1 Introduction

The MixedMedia service is defined to support hardware devices exists which are able to accept cash and checks in the same transaction either in a single or multiple bunches. The service provides a simple way to report the device's capability to perform such a transaction as well as the methods required to perform such a transaction.

If mixed media is enabled (both cashAccept and checkAccept are *true*), then moving the media can be performed using either *CashAcceptor* or *Check* commands. While equivalent commands such as CashAcceptor.CashInEnd and Check.MediaInEnd perform the same function and could therefore be used interchangeably, it is recommended to use one or other to move the media as this allows existing non mixed media transaction flows to be re-used. The only real advantage to either is that check image parameters can be specified using Check.MediaIn and no such parameters exist in the *CashAcceptor* service, therefore default or pre-configured check image parameters have to apply.

The interface follows these principles:

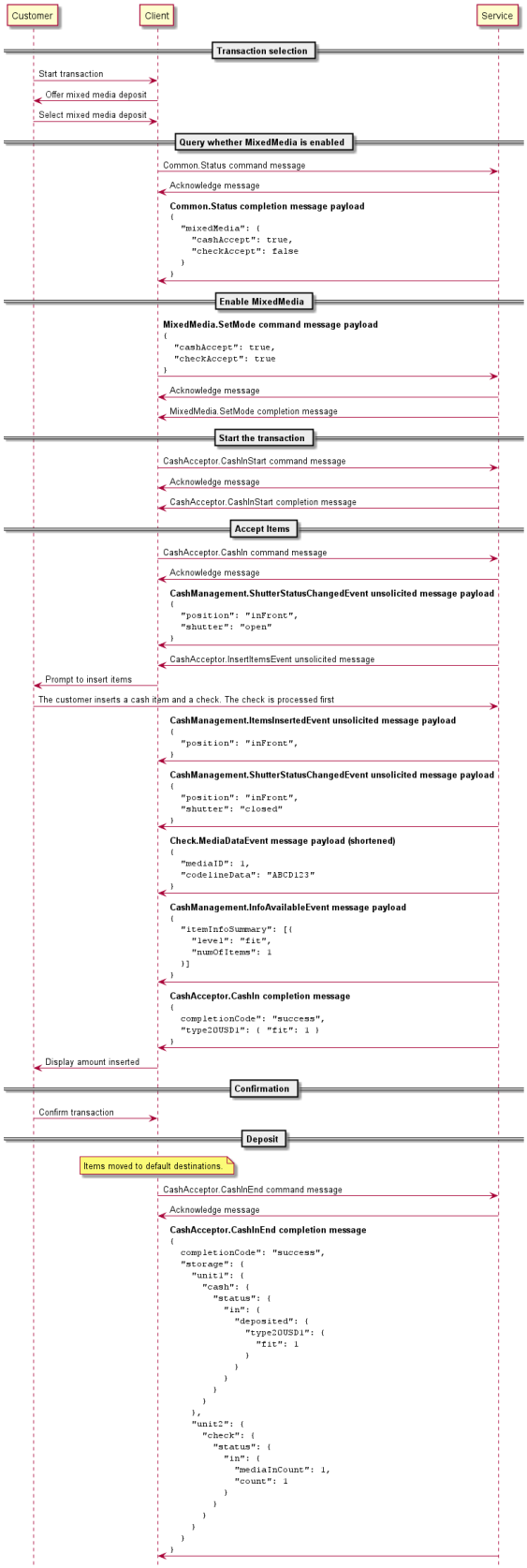
- A transaction is not mixed unless the device is capable of performing such an operation and configured to do so.
- If the transaction is accepted and media is deposited to storage units, the same storage units are used regardless of whether the transaction was performed using *CashAcceptor* or *Check* commands.
- Media acceptance is based on the current configuration appropriate to the media, for example if EUR 500 has been disabled for acceptance using CashAcceptor.ConfigureNoteTypes, then it is also disabled during a mixed media transaction.
- Events appropriate to the media detected are sent, so for example a Check.MediaDataEvent is sent if a check is detected, while cash events such as CashManagement.InfoAvailableEvent are posted if cash items are detected. If a given item is not detected as either media type (for example, not cash and no codeline detected), it can be reported using either interface.
- If supported, the type of transaction can be changed using the MixedMedia.SetMode command. Example usage may be that customers of the financial institution may be permitted to perform mixed media transactions while customers of other financial institution may only be permitted to deposit cash. Another example may be that mixed media transactions could be disabled if there is no storage unit available to store checks, allowing check only transactions to return all the media after imaging and endorsement.
- If the media is to be cleared to a single storage unit using a retract or reset command, the storage unit should be capable of accepting all such media. If media is to be sorted to appropriate units, for example by specifying *itemCassette* in CashManagement.Reset, then media are sorted to storage units which support the media types detected.

10.1.2 Example Transaction flows

This section describes some example mixed media transaction flows. In all cases, equivalent functionality can be achieved using CashAcceptor or Check commands, while events appropriate to the items are sent as they are processed. Some of the example flows are implemented using both using interfaces to illustrate this.

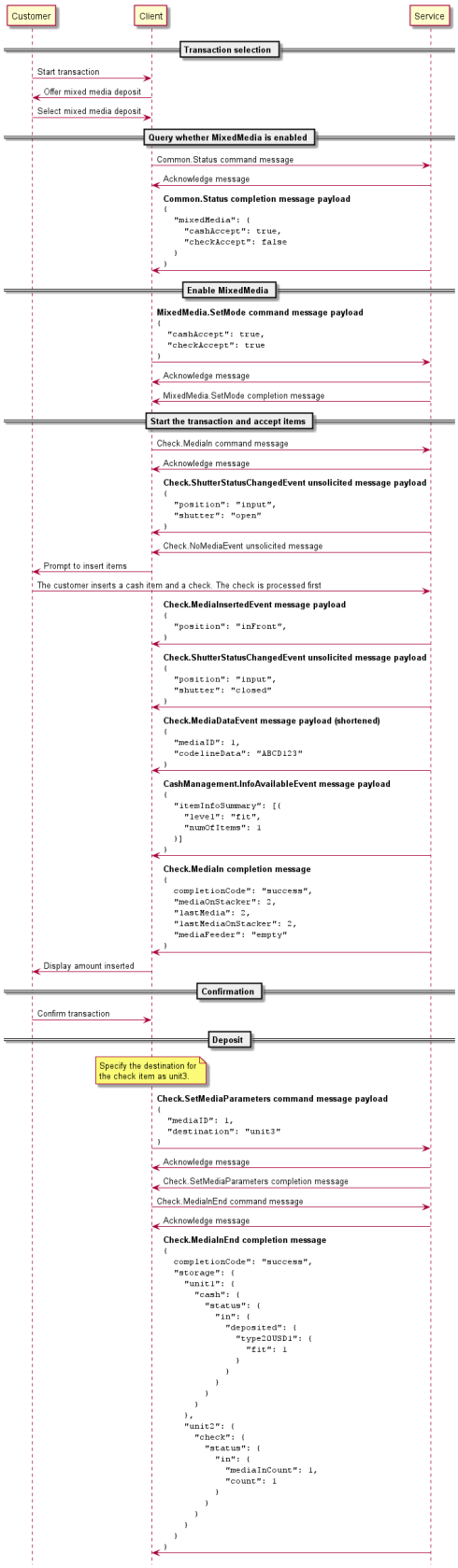
Successful CashAcceptor Mixed Media Transaction

This flow describes a successful MixedMedia transaction flow using CashAcceptor commands to move the media. Successful Check Mixed Media Transaction shows the equivalent flow using Check commands to move the media. The service supports the MixedMedia interface and a mixed media transction can be offered to this customer.



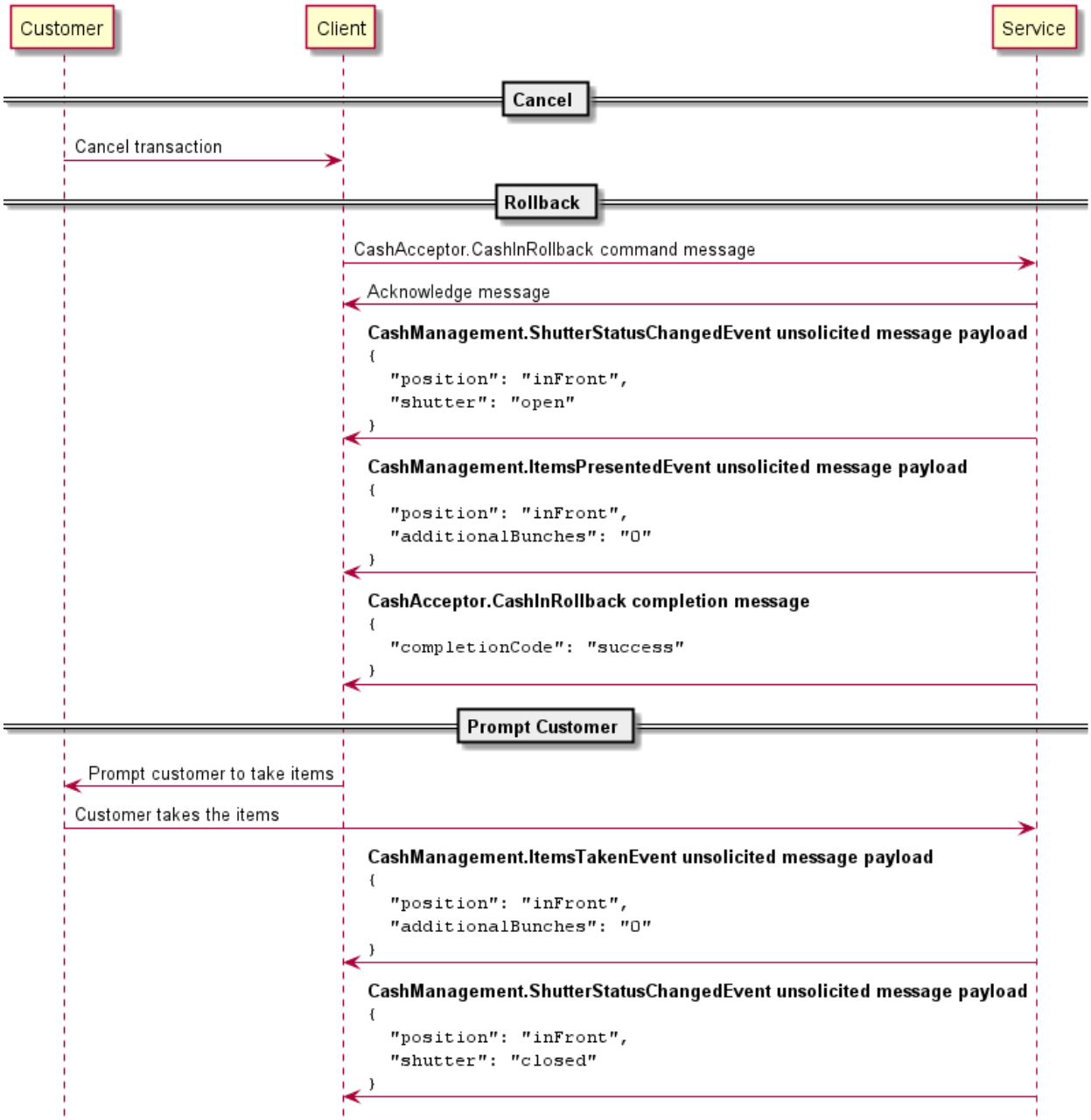
Successful Check Mixed Media Transaction

This flow describes a successful MixedMedia transaction flow using Check commands to move the media. Successful CashAcceptor Mixed Media Transaction shows the equivalent flow using CashAcceptor commands to move the media. The service supports the MixedMedia interface and a mixed media transction can be offered to this customer.



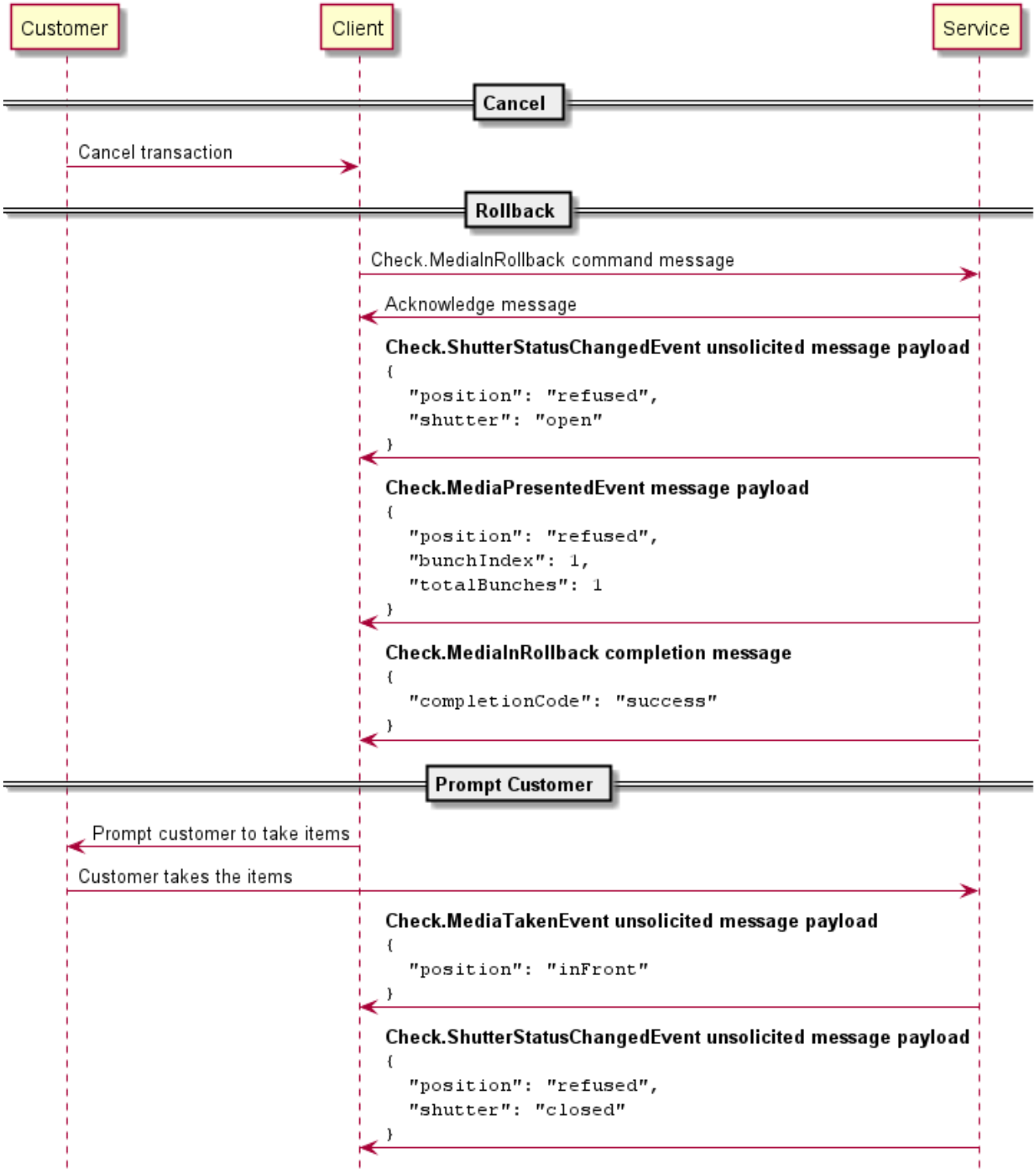
Canceled CashAcceptor Mixed Media Transaction

This flow describes a MixedMedia transaction flow using CashAcceptor commands to move the media, where the transaction is canceled by the application or customer and the media is therefore rolled back. The flow is identical to Successful CashAcceptor Mixed Media Transaction up to *Display amount inserted*.



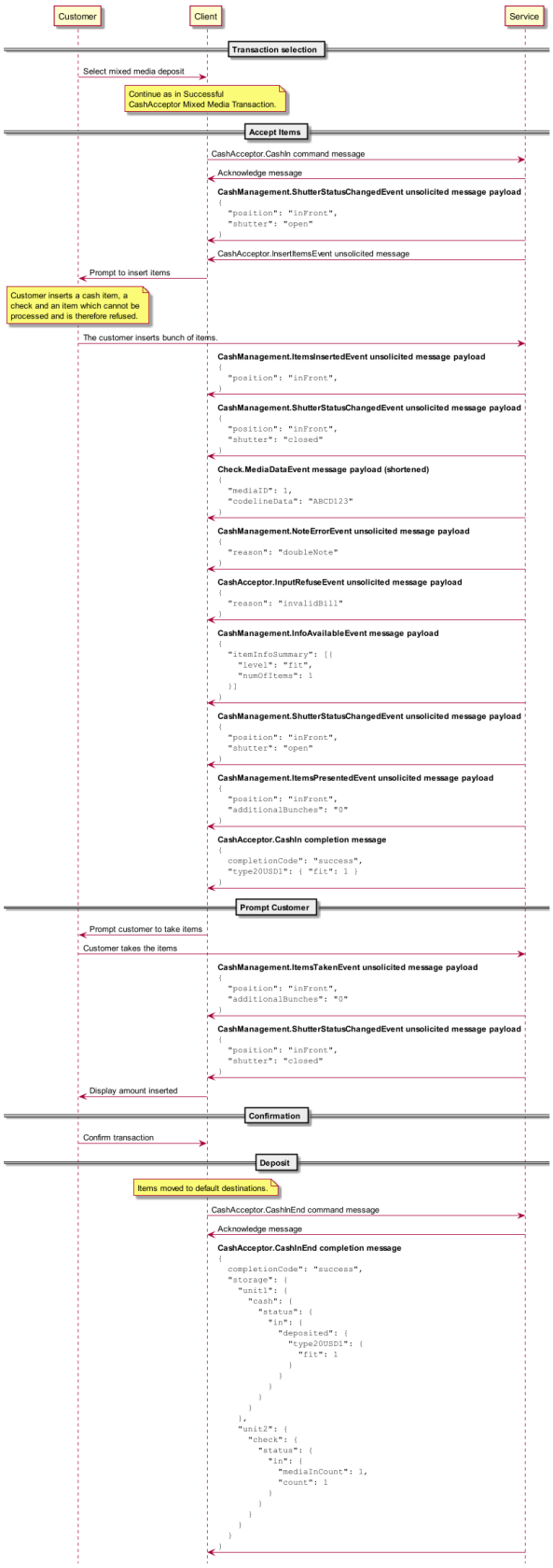
Canceled Check Mixed Media Transaction

This flow describes a MixedMedia transaction flow using Check commands to move the media, where the transaction is canceled by the application or customer and the media is therefore rolled back. The flow is identical to Successful Check Mixed Media Transaction up to *Display amount inserted*.



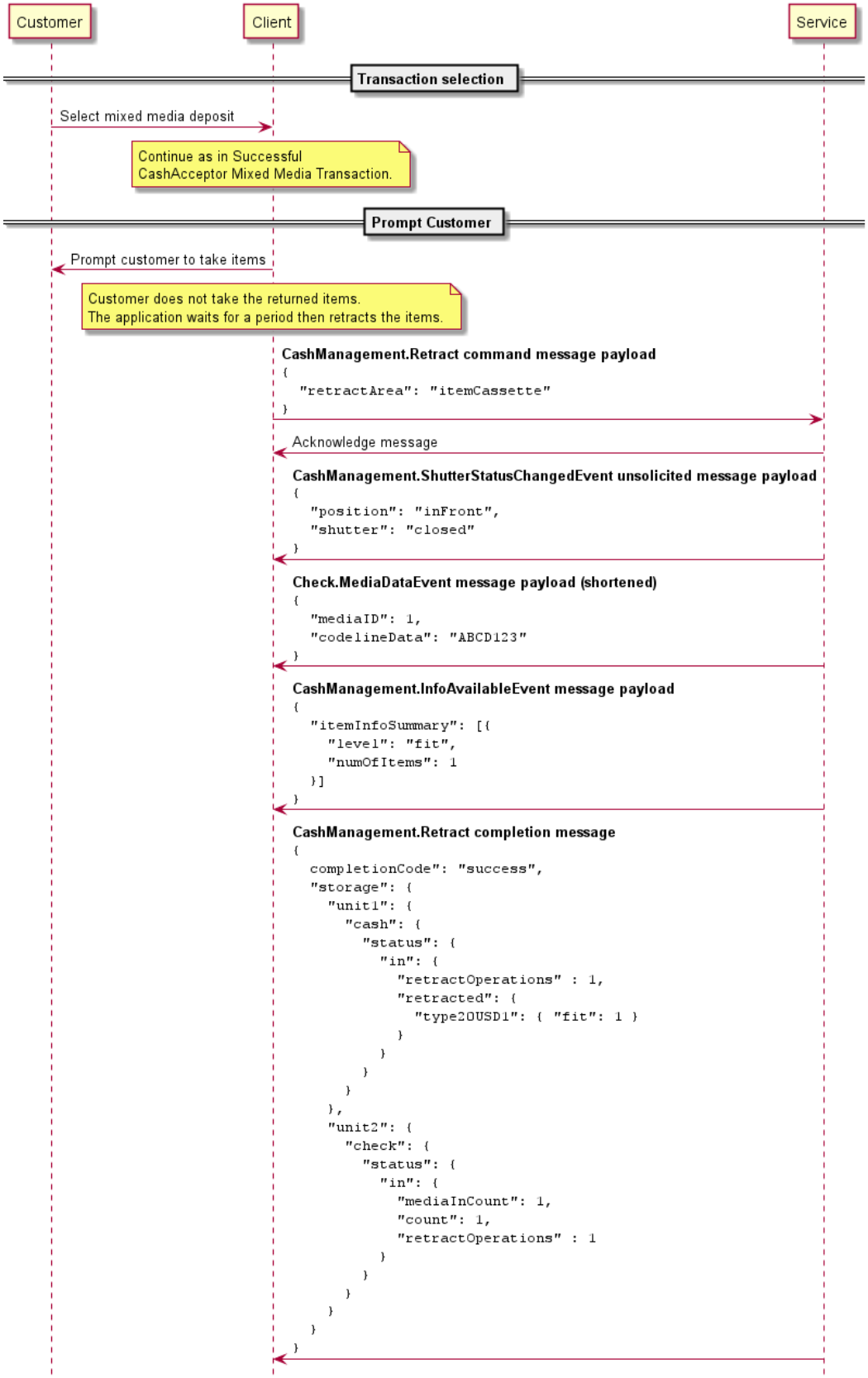
Successful CashAcceptor Mixed Media Transaction with item(s) refused

This flow describes a successful MixedMedia transaction flow using CashAcceptor commands to move the media. One item is refused and taken by the customer. The flow is identical to Successful CashAcceptor Mixed Media Transaction up to *Accept Items*. The transaction can also be performed using equivalent Check commands.



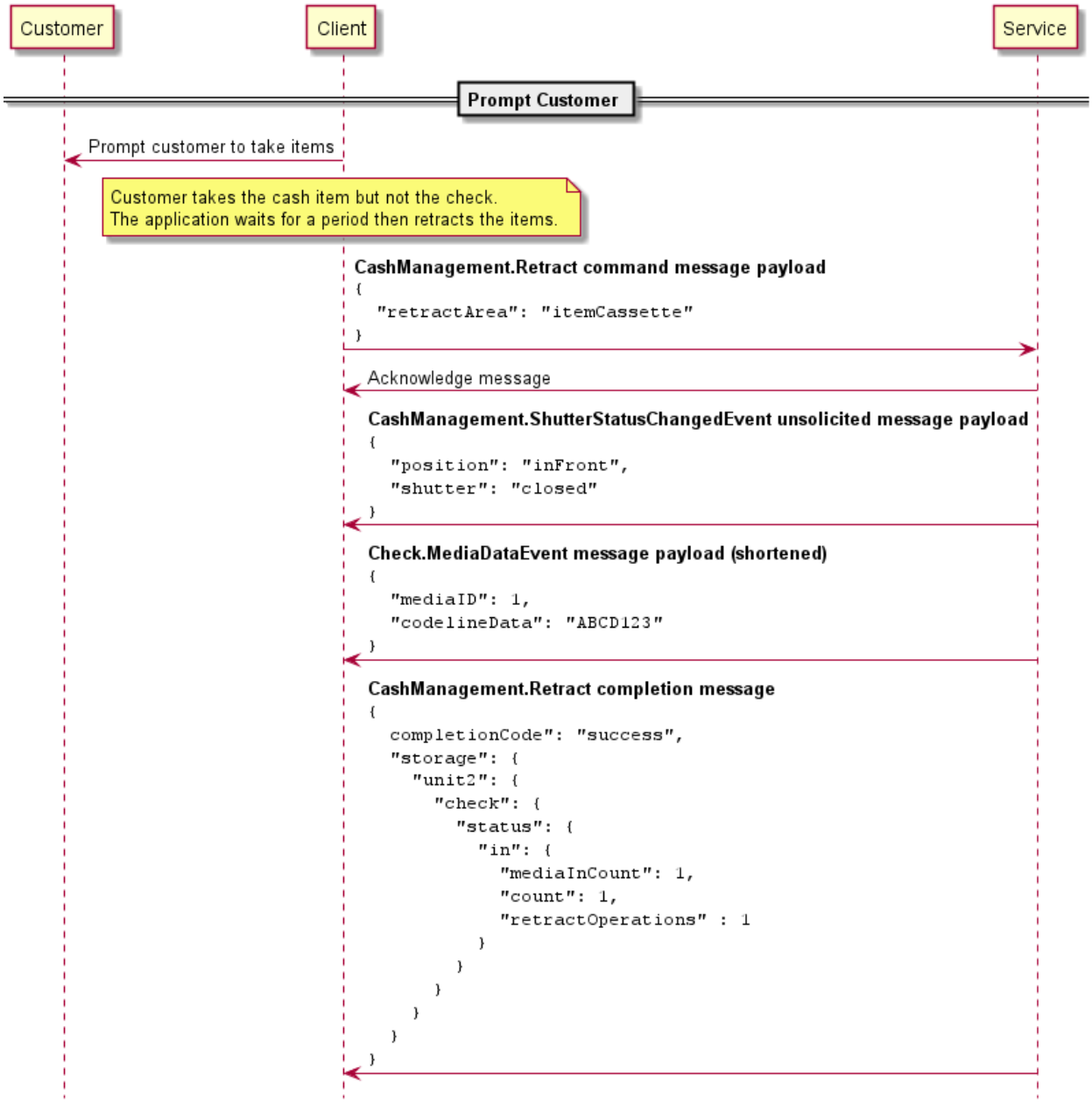
CashAcceptor Mixed Media Transaction with item(s) retracted

This flow describes a MixedMedia transaction flow using CashAcceptor commands to move the media, where the transaction is canceled by the application or customer and the media is therefore rolled back, but the media is not taken by the customer and is therefore retracted after a timeout expires. The same media which was rolled back is retracted. The flow is identical to Successful CashAcceptor Mixed Media Transaction up to *Prompt Customer*.



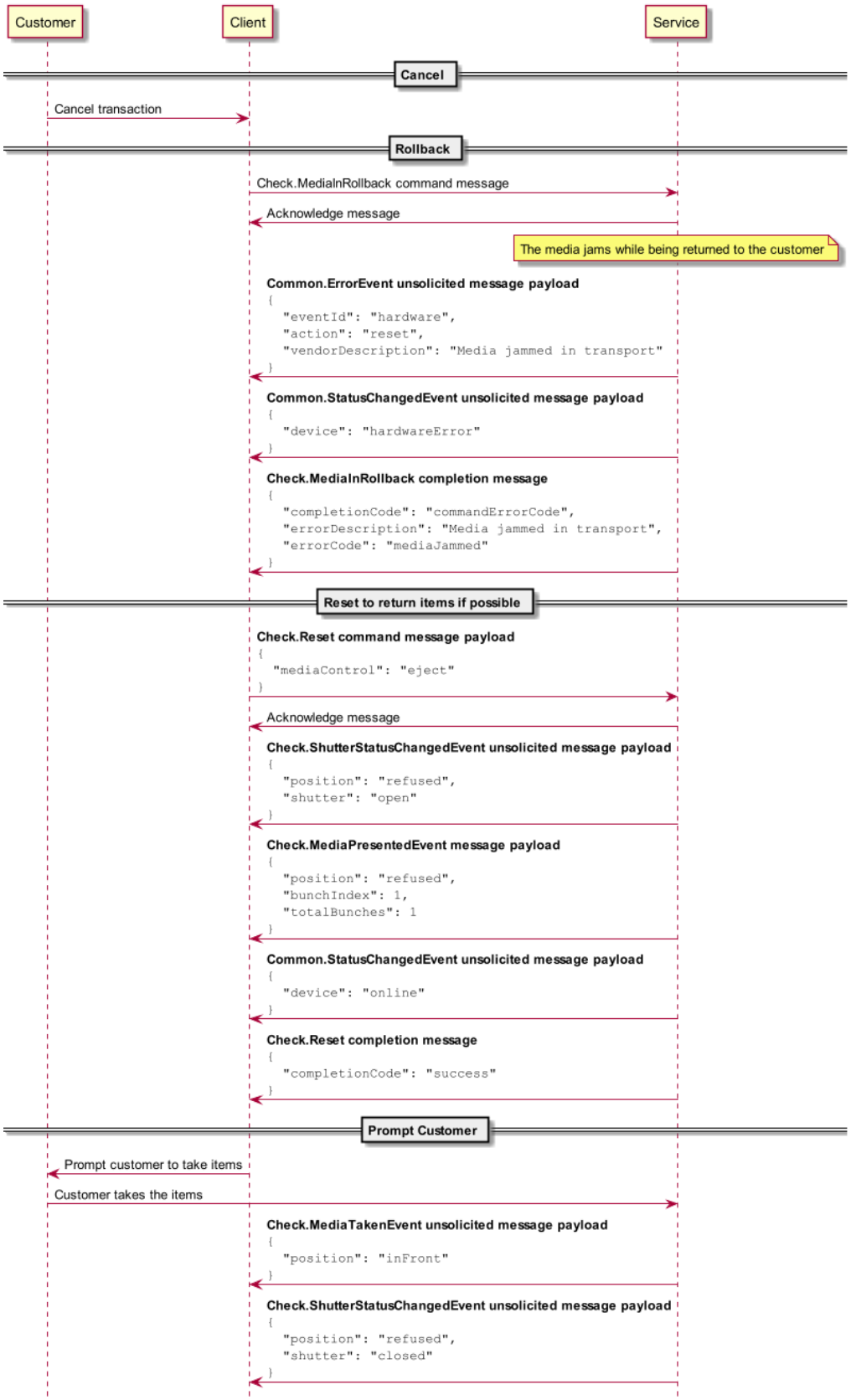
CashAcceptor Mixed Media Transaction with items returned, some taken and others retracted

This flow describes a MixedMedia transaction flow using CashAcceptor commands to move the media, where the transaction is canceled by the application or customer and the media is therefore rolled back. The customer takes one of the items rolled back but not all, therefore the remaining item is retracted after a timeout expires. The flow is identical to Successful CashAcceptor Mixed Media Transaction up to *Prompt Customer*.



Check Mixed Media Transaction where items jam during Rollback

This flow describes a MixedMedia transaction flow using Check commands to move the media. A hardware error occurs during the transaction therefore Check.Reset is used to recover and in this case the media was successfully returned to the customer and taken. The same flow can be performed using equivalent CashAcceptor commands. The flow is identical to Successful Check Mixed Media Transaction up to *Display amount inserted*.


Customer
Client
Service
Cancel
Cancel transaction
Rollback
Check.MediaInRollback command message
Acknowledge message
The media jams while being returned to the customer
Common.ErrorEvent unsolicited message payload
{
"eventId": "hardware",
"action": "reset",
"vendorDescription": "Media jammed in transport"
}
Common.StatusChangedEvent unsolicited message payload
{
"device": "hardwareError"
}
Check.MediaInRollback completion message
{
"completionCode": "commandErrorCode",
"errorDescription": "Media jammed in transport",
"errorCode": "mediaJammed"
}
Reset to return items if possible
Check.Reset command message payload
{
"mediaControl": "eject"
}
Acknowledge message
Check.ShutterStatusChangedEvent unsolicited message payload
{
"position": "refused",
"shutter": "open"
}
Check.MediaPresentedEvent message payload
{
"position": "refused",
"bunchIndex": 1,
"totalBunches": 1
}
Common.StatusChangedEvent unsolicited message payload
{
"device": "online"
}
Check.Reset completion message
{
"completionCode": "success"
}
Prompt Customer
Prompt customer to take items
Customer takes the items
Check.MediaTakenEvent unsolicited message payload
{
"position": "inFront"
}
Check.ShutterStatusChangedEvent unsolicited message payload
{
"position": "refused",
"shutter": "closed"
}

10.2 Command Messages

10.2.1 MixedMedia.SetMode

This command is used to set the transaction mode for the device and is only applicable for MixedMedia processing. The mode determines which type of item the device will process in subsequent transactions.

An example of how this can be used is on a device which can accept cash and checks. The decision whether to allow acceptance of either or both types of media could be based on the individual customer, or switched dynamically based on bank or local requirements.

The mode:

- Applies to all subsequent transactions on this device
- Is persistent
- Is unaffected by a device reset by for example a Check.Reset or reset on another interface
- Fails if a transaction is in progress on the device if the dynamic property is false
- Fails with the *invalidData* error where an attempt is made to set a mode that is not supported.

The current mode is reported by mixedMedia status. If the command is successful, status is updated.

Command Message

Payload (version 2.0)	Type	Required
{		
"modes": {	object	✓
"cashAccept": true,	boolean, null	
"checkAccept": true	boolean, null	
}		
}		

Properties

modes
Specifies the required mixed media modes.
Property value constraints:
`minProperties: 1`

modes/cashAccept
Specifies whether transactions can accept cash. This property may be null if no change required or its state has not changed in Common.StatusChangedEvent.
default: null

modes/checkAccept
Specifies whether transactions can accept checks. This property may be null if no change required or its state has not changed in Common.StatusChangedEvent.
default: null

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "transactionActive"	string, null	
}		

Properties
errorCode Specifies the error code if applicable, otherwise null. Following values are possible: • `transactionActive` - A transaction is active and dynamic is false. default: null

Event Messages

None

11. 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.

11.1 General Information

11.1.1 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.

11.1.2 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.

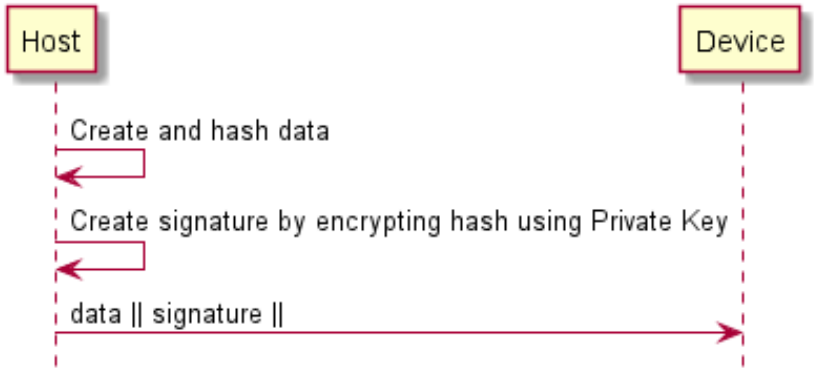
Definitions and Abbreviations	**Description**
ATM	Automated Teller Machine, used here for any type of self-service terminal, regardless whether it actually dispenses cash.
CA	Certificate Authority.
Certificate	A data structure that contains a public key and a name that allows certification of a public 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.

11.1.3 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}.

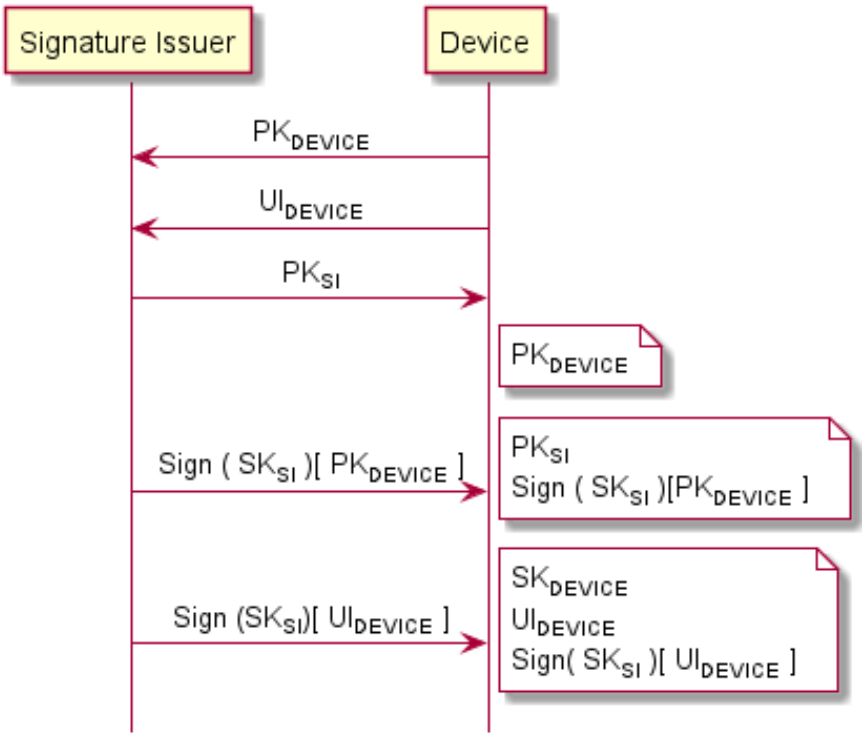
The Device will then validate the signature using PK_{HOST}, verify the random number and then obtain the master key by decrypting using SK_{DEVICE}.

The commands used to exchange master symmetric keys as described above are:

- KeyManagement.StartKeyExchage
- KeyManagement.ImportKey

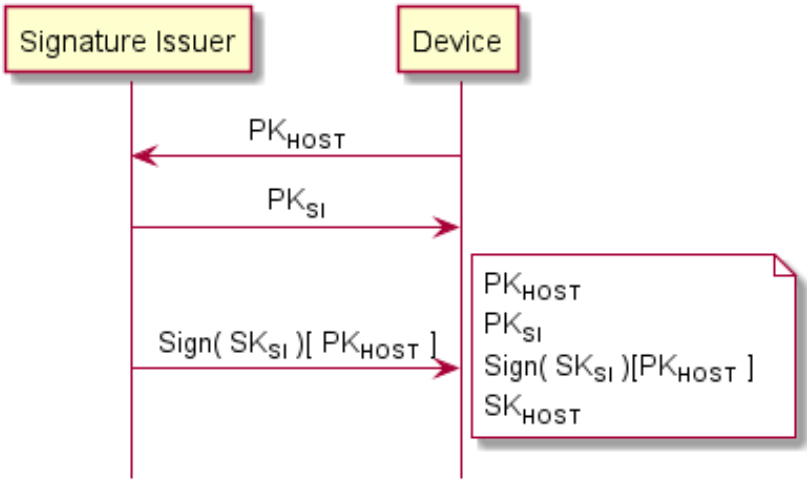
Initialization Phase – Signature Issuer and ATM PIN

This would typically occur in a secure manufacturing environment.



Initialization Phase – Signature Issuer and Host

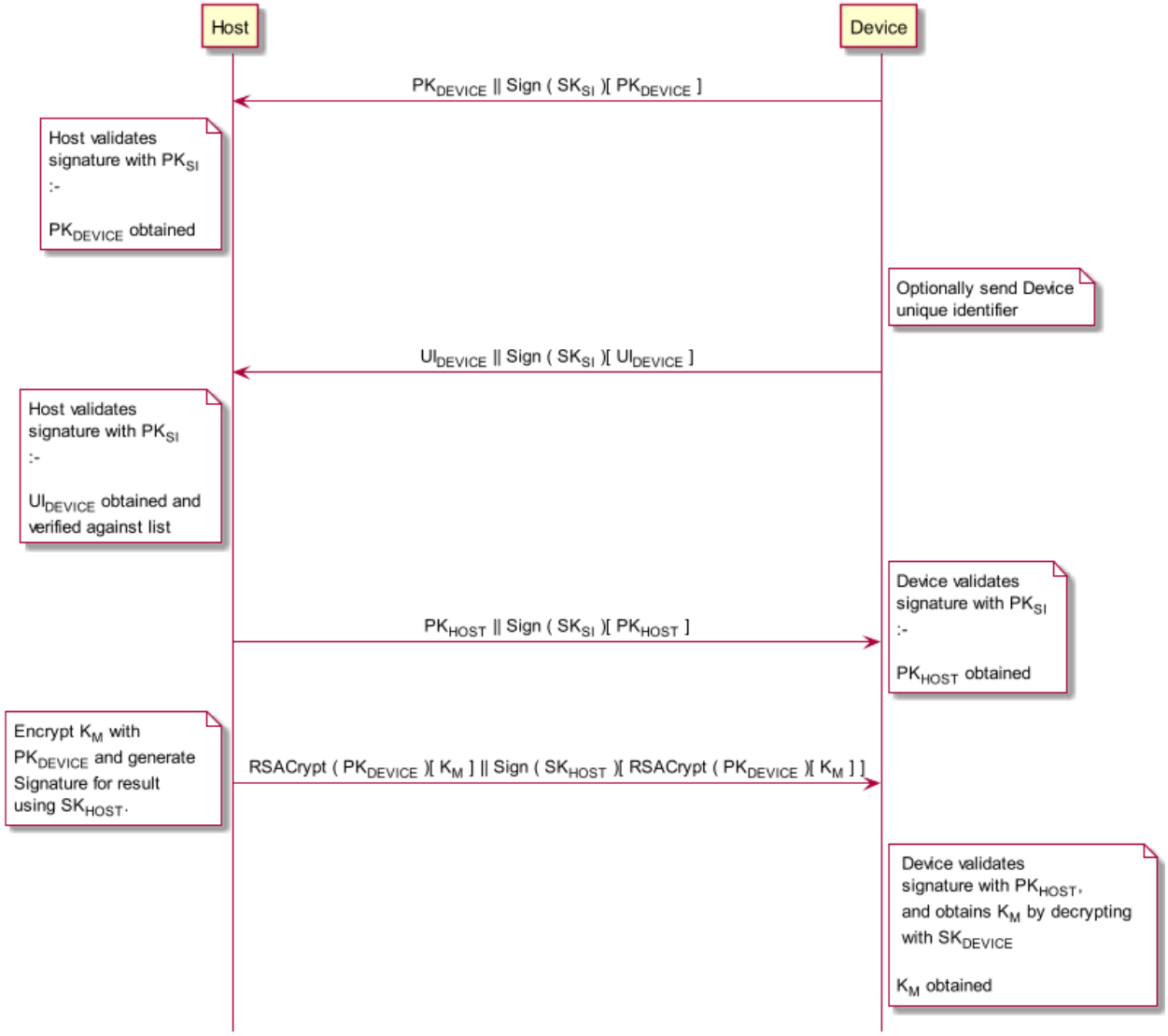
This would typically occur in a secure offline environment.



Key Exchange – Host and ATM PIN

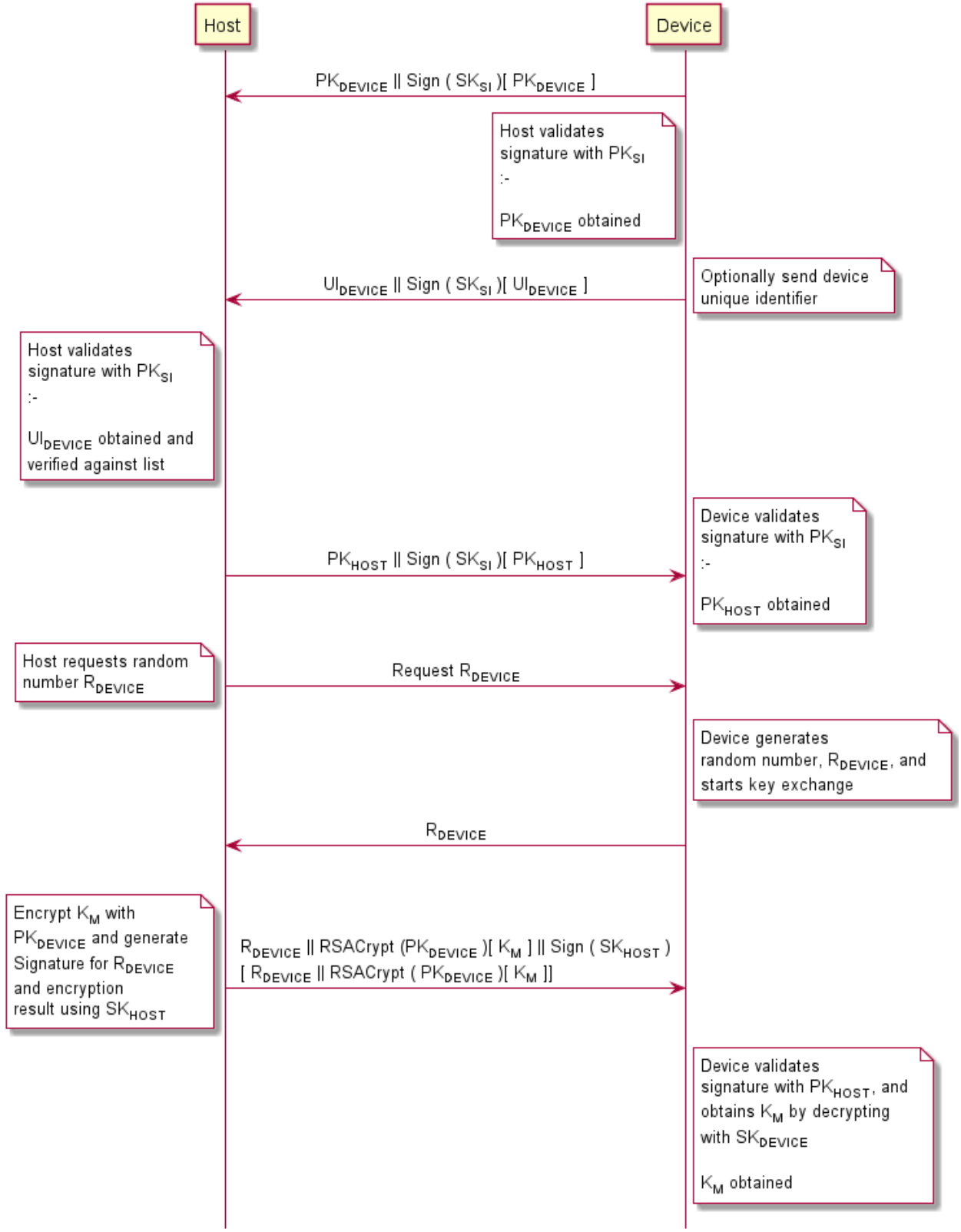
This following is a typical interaction for the exchange of the initial symmetric master key between host and device.

The following is the recommended sequence of interchanges.



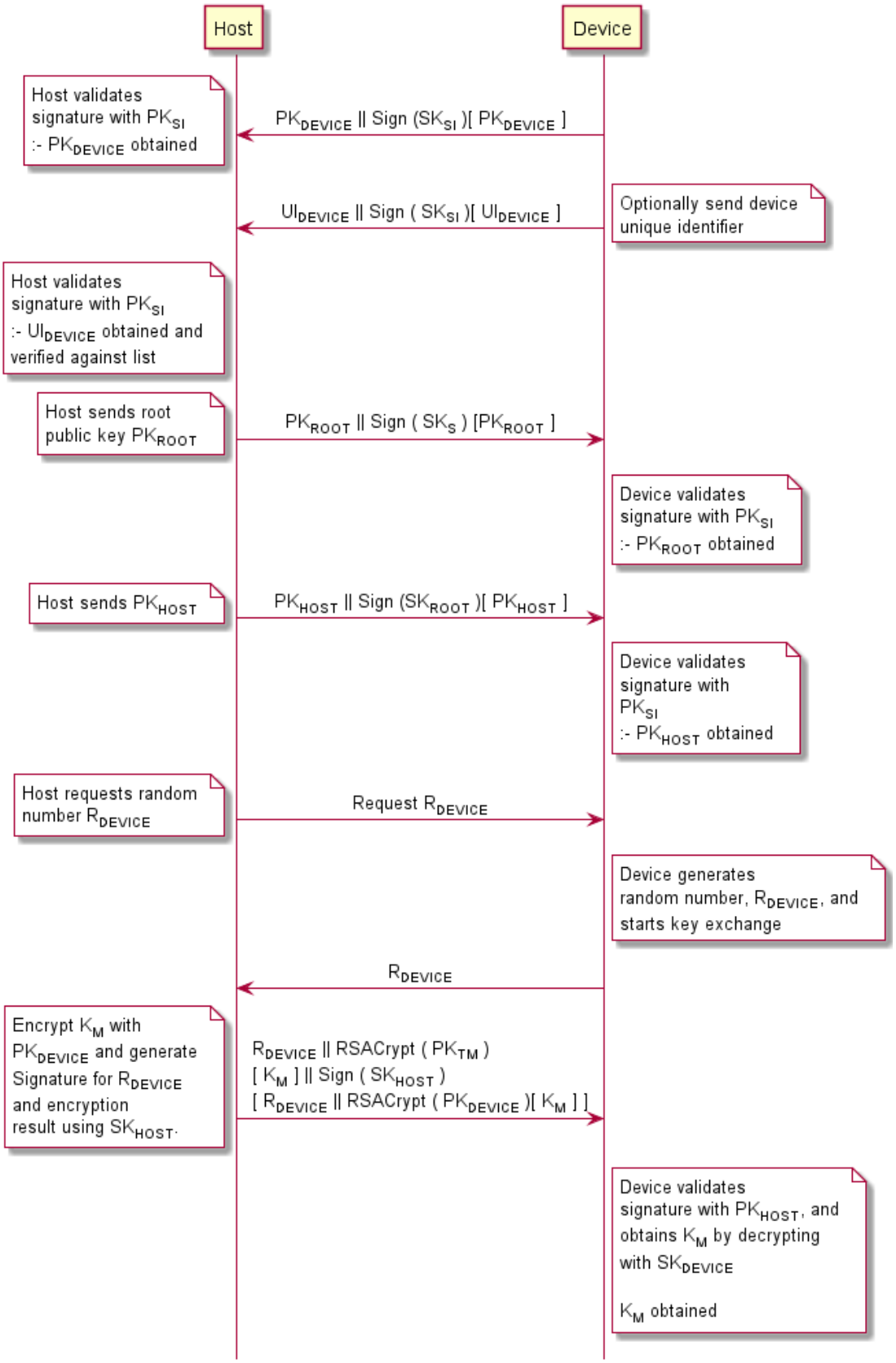
Key Exchange (with random number) – Host and ATM PIN

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.

11.1.4 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.

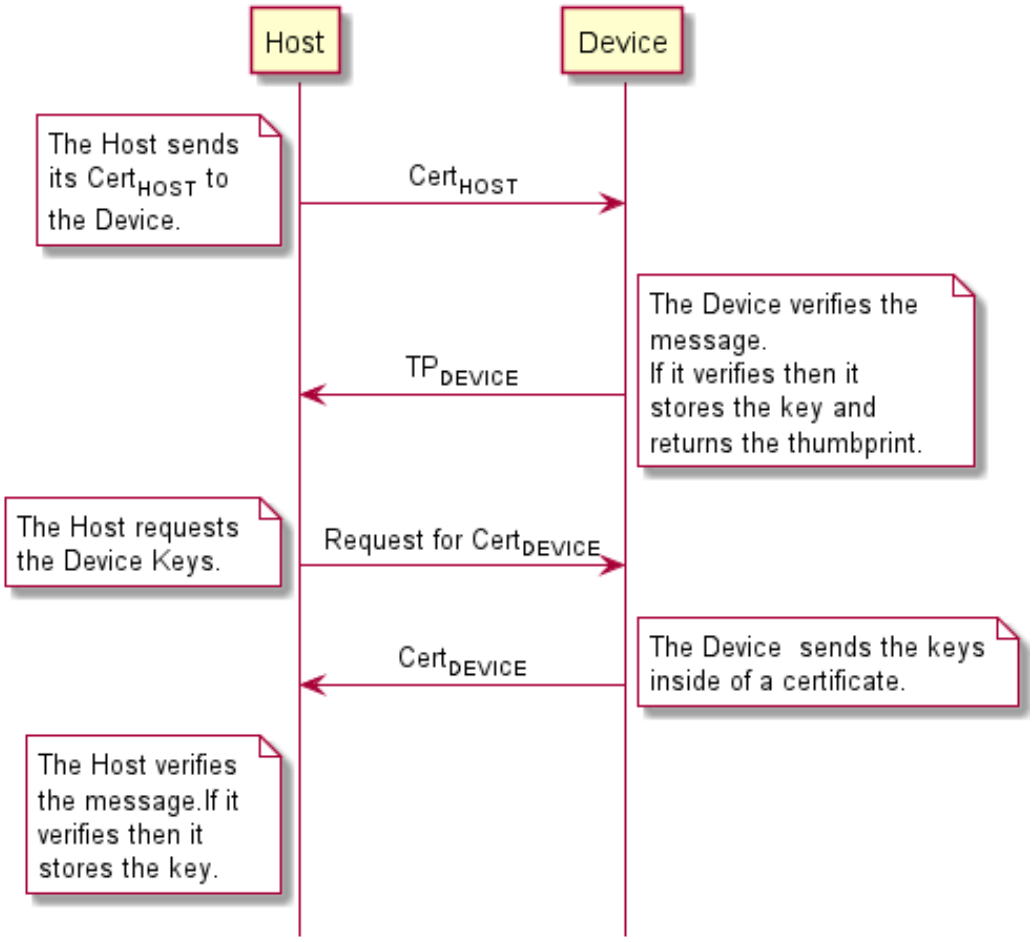
Certificate Exchange and Authentication

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:

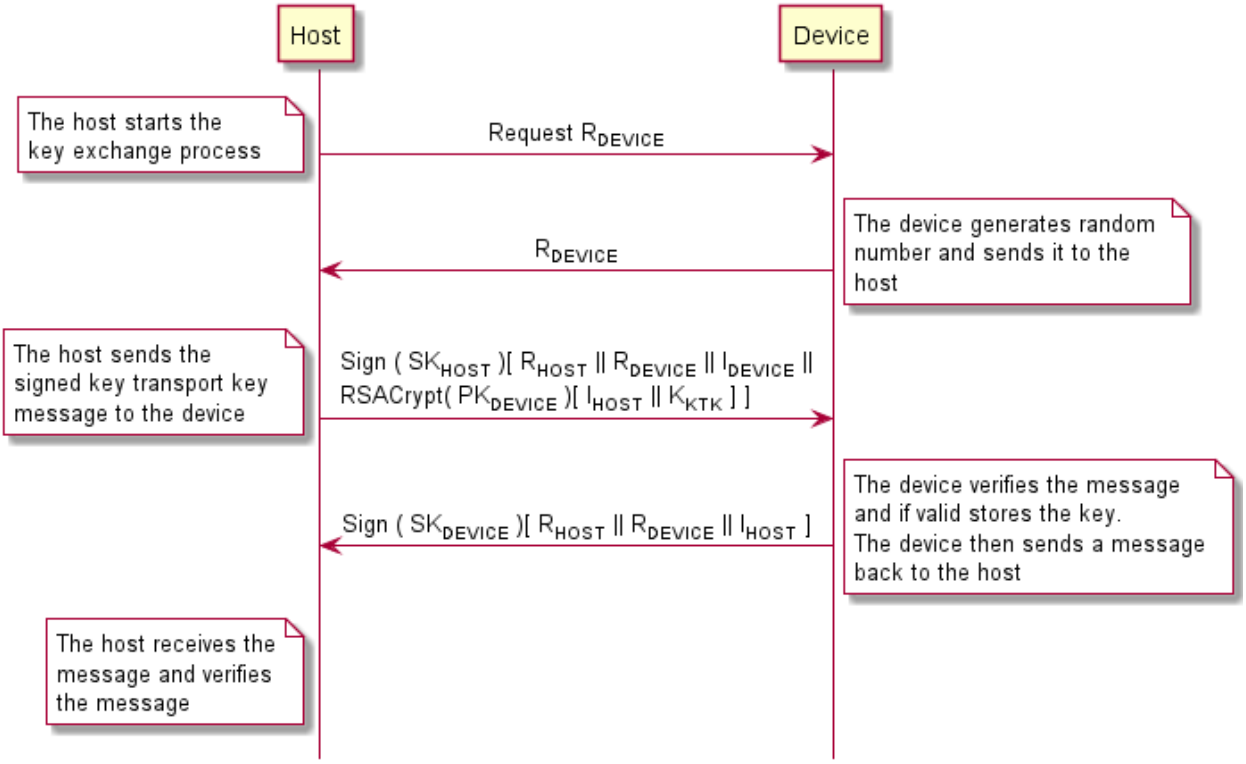


Remote Key Exchange

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.

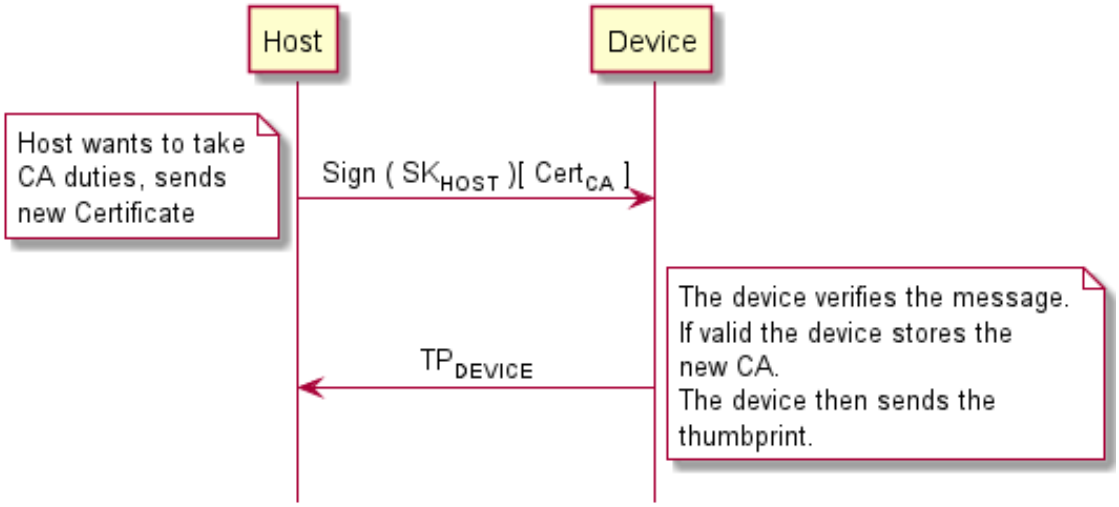
The following diagram below shows how the host and device transmit the Key Transport Key.



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.

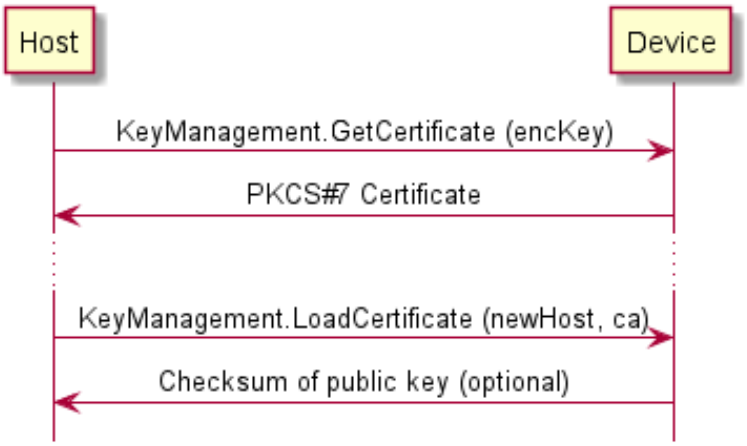
After the host tells the device to shift to the secondary certificate state, only Secondary Certificates can be used. The device will no longer be able to go back to the Primary State and any attempts from the host to get or load a Primary Certificate will return an error. When either Primary or Secondary certificates are compromised it is up to the vendor on how the device should be handled with the manufacturer.

11.1.5 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

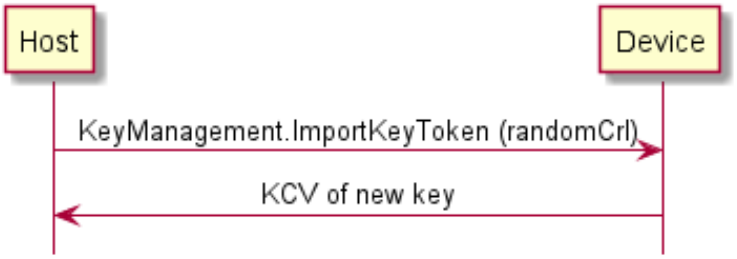
There are two protocols that can be used to transport symmetric keys under TR34; these are the One Pass and Two Pass protocols. The use of XFS4IoT commands for these two protocols are shown in the following sections.

- 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].

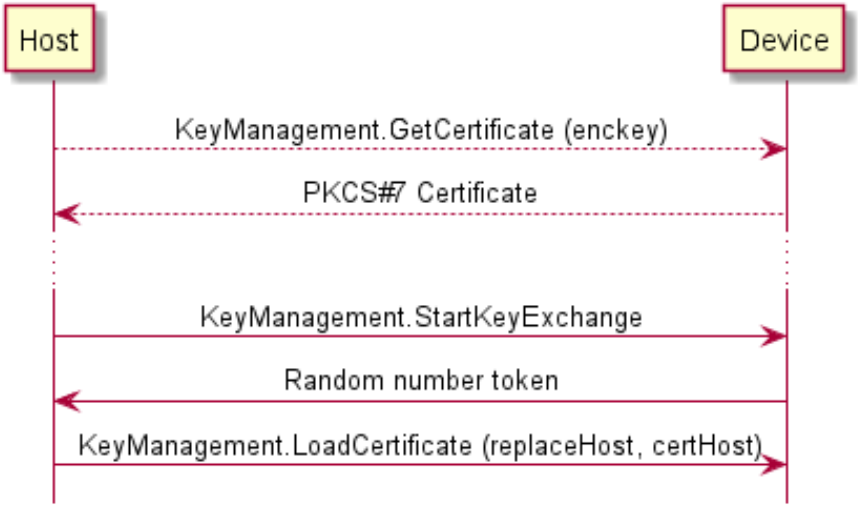
Pre-condition: A successful BIND command has completed such that the device is bound to the host.

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.

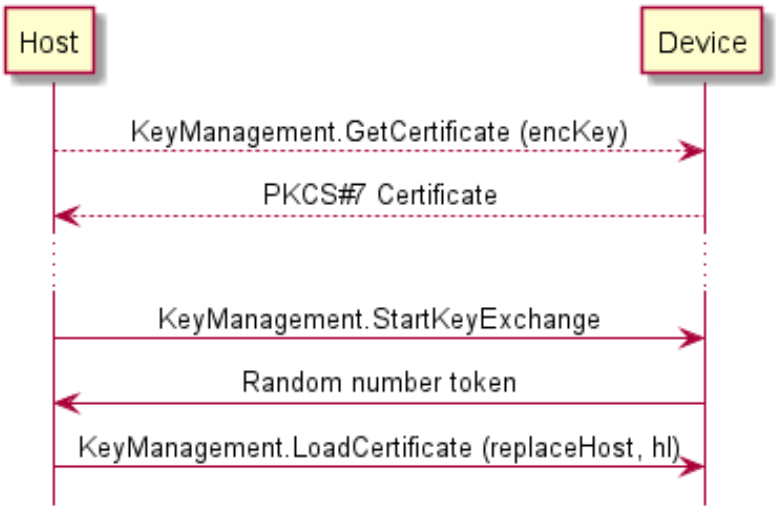


NB: Dotted lines represent commands that are only required if the device encryption certificate has not been previously stored by the host.

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.



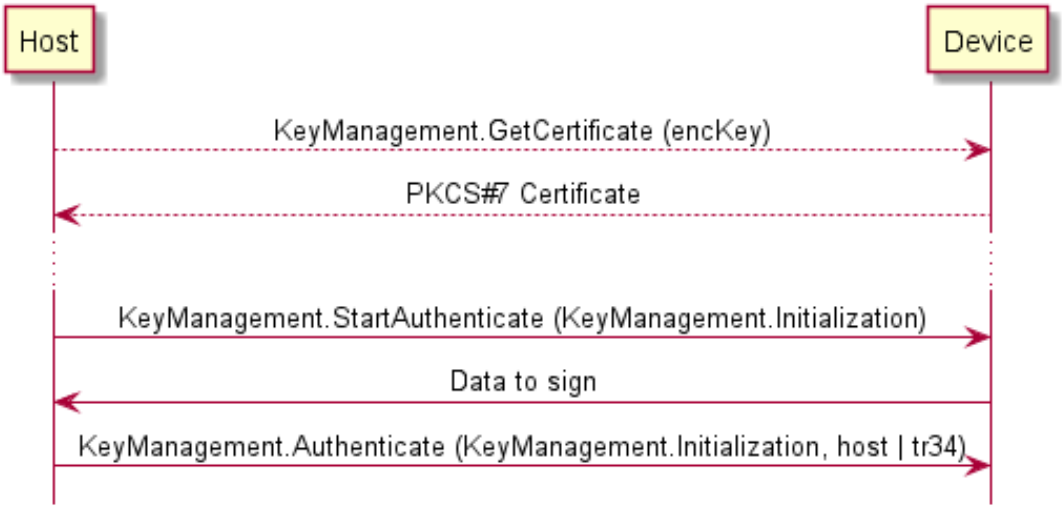
NB: Dotted lines represent commands that are only required if the device encryption certificate has not been previously stored by the host.

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.

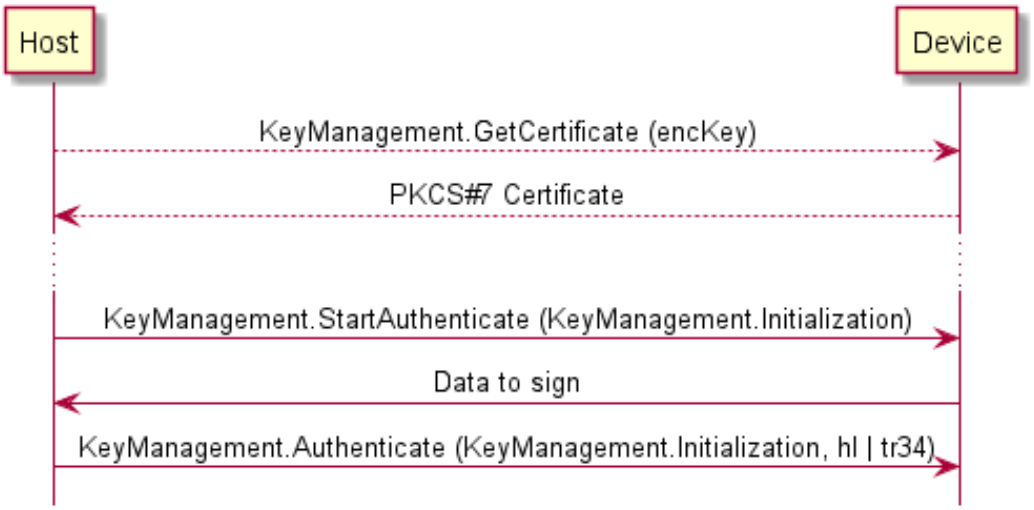


NB: Dotted lines represent commands that are only required if the device encryption certificate has not been previously stored by the host.

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.

11.1.6 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
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.

11.1.7 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
  }
}
```

11.1.8 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.

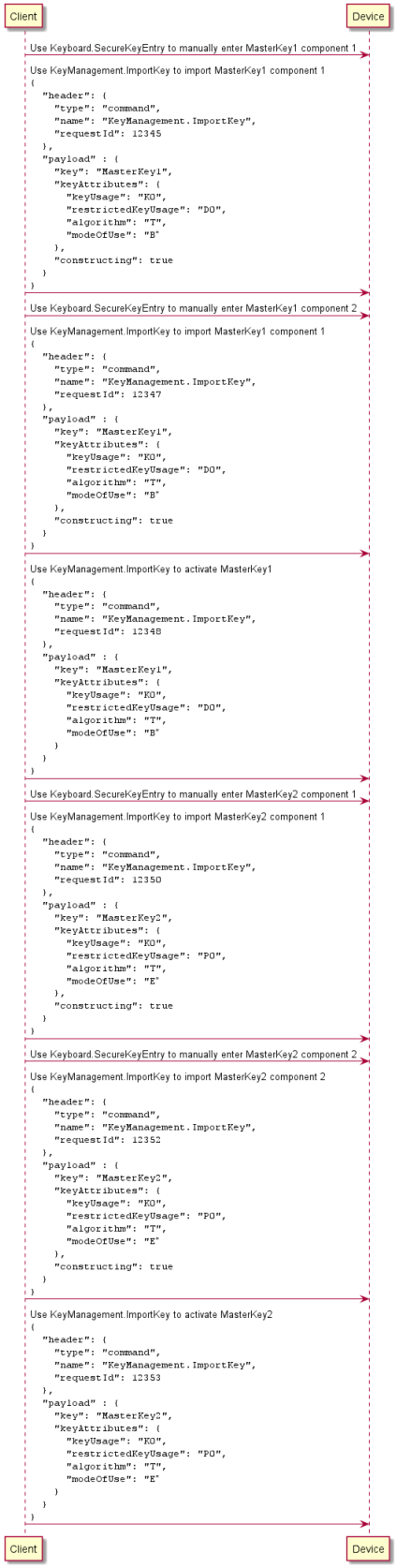
If DUKPT is supported, this key must be included in the KeyManagement.GetKeyDetail output.

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).

11.1.9 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:

11.1.10 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

11.2 Command Messages

11.2.1 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 2.0)	Type	Required
{		
"keyName": "Key01"	string, null	
}		

Properties

keyName

Name of the key for which detailed information is requested. If this property is null, detailed information about all the keys in the encryption module is returned.

default: null

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "keyNotFound",	string, null	
"keyDetails": {	object, null	
"*exampleProperty1*": {	object	
"generation": 0,	integer, null	
"version": 0,	integer, null	
"activatingDate": "20210101",	string, null	
"expiryDate": "20220101",	string, null	
"loaded": "no",	string	✓
"keyBlockInfo": {	object	✓
"keyUsage": "K0",	string	✓
"restrictedKeyUsage": "D0",	string, null	
"algorithm": "T",	string	✓
"modeOfUse": "B",	string	✓
"keyVersionNumber": "01",	string, null	
"exportability": "N",	string	✓
"optionalBlockHeader": "HM0621",	string, null	
"keyLength": 0	integer	

Payload (version 2.0)	Type	Required
}		
},		
"*exampleProperty2*": See *keyDetails/exampleProperty1* properties	object	
}		
}		

Properties

errorCode

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

- `keyNotFound` - The specified key name is not found.

default: null

keyDetails

This property contains key/value pairs where the key is a name of key and the value is the key detail. If there are no key details, this property is an empty object.

default: null

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 null if no such information is available for the key.

Property value constraints:

```
minimum: 0
maximum: 99
```

default: null

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 null if no such information is available for the key.

Property value constraints:

```
minimum: 0
maximum: 99
```

default: null

keyDetails/exampleProperty1/activatingDate

Specifies the date when the key is activated in the format YYYYMMDD. This value can be null 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])$
```

default: null

keyDetails/exampleProperty1/expiryDate

Specifies the date when the key expires in the format YYYYMMDD. This value can be null 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])$
```

default: null

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 value can be null 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]$
```

default: null

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 null if Key versioning is not used.

Property value constraints:

```
pattern: ^[0-9a-zA-Z][0-9a-zA-Z]$
```

default: null

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 null if there are no optional block headers.

default: null

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` default: 0

Event Messages

None

11.2.2 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 2.0)	Type	Required
{		
"authentication": {	object, null	
"method": "certhost",	string	✓
"key": "Key01",	string, null	
"data": "QXV0aGVudGljYXRpb25EYXRh"	string	✓
}		
}		

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.
- Commands which do not clear or modify the authentication data from the device may be executed between the *KeyManagement.StartAuthenticate* and the authenticated 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.

default: null

Properties
authentication/method Specifies the method used to generate the authentication data. The possible values are: • `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. • `certHostTr34` - The data is signed by the current Host, using TR-34. • `caTr34` - The data is signed by the Certificate Authority (CA), using TR-34. • `hlTr34` - The data is signed by the Higher Level (HL) Authority, using TR-34. • `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 null, the default Signature Issuer or if null, the default Signature Issuer public key (installed in a secure environment during manufacture) will be used. default: null

authentication/data
This property contains the authenticated data (MAC, Signature) generated from 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 *cbcmac* or *cmac* is specified in the *method* property, then *key* must refer to a key with a MAC key usage key e.g. M0.
If *method* is none, this property is not required.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "accessDenied"`	string, null	
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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 *authentication/data* does not match the one previously provided by the device. • `keyNoValue` - A required key was not specified in *authentication.key*. • `keyNotFound` - The key specified in *authentication.key* was not found. • `useViolation` - The key specified in *authentication.key* can not be used for the specified *authentication.method*. • `modeOfUseNotSupported` - The key specified in *authentication.key* can not be used for authentication. • `macInvalid` - The MAC included in *authentication/data* is invalid. • `signatureInvalid` - The signature included in *authentication/data* is invalid. default: null

Event Messages

None

11.2.3 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 2.0)	Type	Required
{		
"derivationAlgorithm": "chipZka",	string	✓
"key": "Key01",	string	✓
"keyGenKey": "Key02",	string	✓
"storedKey": "StoredIVKey",	string, null	
"iv": {	object, null	
"key": "KeyToDecrypt",	string, null	
"value": "VGhlIGluaXRpYWxpemF0 ..."	string	✓
},		
"padding": 255,	integer	
"inputData": "a2V5IGRlcml2YXRpb24g ..."	string	✓
}		

Properties

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.

storedKey
This specifies the name of a key (usage 'I0') used as the Initialization Vector (IV). This property is null if not required.
default: null

iv
Specifies the Initialization Vector. This property is null if *storedKey* is used.
default: null

iv/key
The name of a key used to decrypt the *value*. This specifies the name of a key (usage 'K0') used to decrypt the *value*. This is only used when the *key* usage is 'D0' and *cryptoMethod* is either CBC or CFB. if this property is null, *value* is used as the Initialization Vector.
default: null

iv/value
The plaintext or encrypted IV for use with the CBC or CFB encryption methods.
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`
default: 0

inputData
Data to be used for key derivation.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "accessDenied"`	string, null	
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null

Event Messages

None

11.2.4 KeyManagement.Reset

Sends a service reset to the Service.

Command Message

Payload (version 2.0)
This message does not define any properties.

Completion Message

Payload (version 2.0)
This message does not define any properties.

Event Messages

None

11.2.5 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 2.0)	Type	Required
{		
"key": "Key01",	string	✓
"keyAttributes": {	object, null	
"keyUsage": "P0",	string	✓
"algorithm": "T",	string	✓
"modeOfUse": "G",	string	✓
"restrictedKeyUsage": "K0"	string	
},		
"value": "a2V5IHZhbHVl",	string, null	
"constructing": false,	boolean	
"decryptKey": "Key01",	string, null	
"decryptMethod": "ecb",	string, null	
"verificationData": "ZGF0YSB0byBiZSB2ZXJp ...",	string, null	
"verifyKey": "VerifyKey01",	string, null	
"verifyAttributes": {	object, null	
"cryptoMethod": "kcvNone",	string, null	
"hashAlgorithm": "sha1"	string, null	
},		
"vendorAttributes": "See vendor documentation"	string, null	
}		

Properties

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. If a keyblock is being imported, this property can be null.
default: null

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 null.

Property value constraints:

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

default: null

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 null if the data in *value* is not encrypted or the *constructing* property is true.

default: null

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 should be null if the *constructing* property is true or if *decryptKey* is null.

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 null. X9.143 defines the cryptographic methods used for each key block version.

default: null

verificationData

Contains the data to be verified before importing.

This property can be null 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
```

default: null

verifyKey

Specifies the name of the previously loaded key which will be used to verify the *verificationData*. This property can be null when no verification is needed before importing the key or the *constructing* property is true.

default: null

Properties
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 null if *verificationData* is not required or the *constructing* property is true. default: null
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 null. 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. default: null
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 null. default: null
vendorAttributes Specifies the vendor attributes of the key to be imported. Refer to vendor documentation for details. default: null

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "keyNotFound",`	string, null	
`"verificationData": "dmVyaWZpY2F0aW9uIGRh ...",`	string, null	
`"verifyAttributes": {`	object, null	
`"keyUsage": "M0",`	string	✓
`"algorithm": "T",`	string	✓
`"modeOfUse": "V",`	string	✓
`"cryptoMethod": "kcvNone",`	string, null	
`"hashAlgorithm": "sha1"`	string, null	
`},`		

Payload (version 2.0)	Type	Required
"keyLength": 0	integer	
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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* or the usage of decryptKey is not supported.
- `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 *verificationData* 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.
- `invalidValue` - The key value contains a key block which failed its authentication check. The key is not stored in the device.
- `formatInvalid` - The format of the key block is invalid.
- `contentInvalid` - The content of the key block is invalid.
- `formatNotSupported` - The key block version or content is not supported.

default: null

verificationData

The verification data.

This property can be null if there is no verification data.

Property value constraints:

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

default: null

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 null if there is no verification data.

default: null

Properties

verifyAttributes/keyUsage

Specifies the key usage. The following values are possible:

- `M0` - ISO 16609 MAC Algorithm 1 (using TDEA).
- `M1` - ISO 9797-1 MAC Algorithm 1.
- `M2` - ISO 9797-1 MAC Algorithm 2.
- `M3` - ISO 9797-1 MAC Algorithm 3.
- `M4` - ISO 9797-1 MAC Algorithm 4.
- `M5` - ISO 9797-1:1999 MAC Algorithm 5.
- `M6` - ISO 9797-1:2011 MAC Algorithm 5 / CMAC.
- `M7` - HMAC.
- `M8` - ISO 9797-1:2011 MAC Algorithm 6.
- `S0` - Asymmetric key pair or digital signature.
- `S1` - Asymmetric key pair, CA key.
- `S2` - Asymmetric key pair, nonX9.24 key.
- `00 – 99` - These numeric values are reserved for proprietary use.

Property value constraints:

```
pattern: ^M[0-8]$|^S[0-2]$|^[0-9][0-9]$
```

verifyAttributes/algorithm

Specifies the encryption algorithm. The following values are possible:

- `A` - AES.
- `D` - DEA.
- `R` - RSA.
- `T` - Triple DEA (also referred to as TDEA).
- `"0" – "9"` - These numeric values are reserved for proprietary use.

Property value constraints:

```
pattern: ^[0-9ADRT]$
```

verifyAttributes/modeOfUse

Specifies the encryption mode. The following values are possible:

- `S` - Signature.
- `V` - Verify Only.
- `0 – 9` - These numeric values are reserved for proprietary use.

Property value constraints:

```
pattern: ^[0-9SV]$
```

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 null.

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.

default: null

Properties
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 null. default: null
keyLength Specifies the length, in bits, of the key. Zero if the key length is unknown. Property value constraints: `minimum: 0` default: 0

Event Messages

None

11.2.6 KeyManagement.DeleteKey

This command deletes a key. If authentication data is required the KeyManagement.StartAuthenticate command should be used to obtain the data to sign.

Deletion of the key may cause other keys to be deleted. On successful completion of this command, it is recommended that clients use the KeyManagement.GetKeyDetail command to check which keys remain loaded.

Command Message

Payload (version 2.0)	Type	Required
{		
"key": "Key01",	string	✓
"authentication": {	object, null	
"method": "certhost",	string	✓
"key": "Key01",	string, null	
"data": "QXV0aGVudGljYXRpb25EYXRh"	string	✓
}		
}		

Properties

key
The name of key being deleted.

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.
- Commands which do not clear or modify the authentication data from the device may be executed between the *KeyManagement.StartAuthenticate* and the authenticated 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.

default: null

authentication/method
Specifies the method used to generate the authentication data. The possible values are:

- `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.
- `certHostTr34` - The data is signed by the current Host, using TR-34.
- `caTr34` - The data is signed by the Certificate Authority (CA), using TR-34.
- `hlTr34` - The data is signed by the Higher Level (HL) Authority, using TR-34.
- `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 null, the default Signature Issuer or if null, the default Signature Issuer public key (installed in a secure environment during manufacture) will be used.
default: null

Properties

authentication/data
This property contains the authenticated data (MAC, Signature) generated from 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 *cbcmac* or *cmac* is specified in the *method* property, then *key* must refer to a key with a MAC key usage key e.g. M0.
If *method* is none, this property is not required.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "accessDenied"`	string, null	
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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 *authentication/data* does not match the one previously provided by the device. • `keyNoValue` - A required key was not specified in *authentication/key*. • `keyNotFound` - The key specified in *authentication/key* was not found. • `useViolation` - The key specified in *authentication/key* can not be used for the specified *authentication/method*. • `modeOfUseNotSupported` - The key specified in *authentication/key* can not be used for authentication. • `macInvalid` - The MAC included in *authentication/data* is invalid. • `signatureInvalid` - The signature included in *authentication/data* is invalid. default: null

Event Messages

None

11.2.7 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 2.0)	Type	Required
{		
"exportItemType": "deviceId",	string	
"name": "PKey01"	string, null	
}		

Properties

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.

default: "deviceId"

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 manufacture for a definition of these default keys. If this is null, then the default EPP public key that is used for symmetric key encryption is exported.

default: null

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "noRSAKeyPair",	string, null	
"value": "aXRlbSBkYXRhIHJlcXVl ...",	string, null	
"rsaSignatureAlgorithm": "na",	string	
"signature": "U2lnbmF0dXJlIGRhdGE="	string, null	
}		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null

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`
default: null

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. default: "na"

signature
The RSA signature of the data item exported.
This should be null when the key signature is not supported.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`
default: null

Event Messages

None

11.2.8 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 2.0)	Type	Required
{		
"key": "Key02",	string	✓
"use": "rsaPrivate",	string	✓
"modulusLength": 0,	integer	
"exponentValue": "device"	string	
}		

Properties

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`
default: 0

exponentValue
Specifies the value of the exponent of the RSA key pair to be generated. The following values are possible:

- `device` - The device will decide the exponent.
- `exponent1` - Exponent of 2^1+1 (3).
- `exponent4` - Exponent of 2^4+1 (17).
- `exponent16` - Exponent of $2^{16}+1$ (65537).

default: "device"

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "accessDenied"	string, null	
}		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null

Event Messages

None

11.2.9 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 2.0)	Type	Required
{		
"exportItemType": "deviceId",	string	
"name": "PKey01",	string	✓
"sigKey": "SigKey01",	string	✓
"signatureAlgorithm": "na"	string	
}		

Properties

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.

default: "deviceId"

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.

default: "na"

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "noRSAKeyPair",	string, null	
"value": "aXRlbSBkYXRhIHJlcXVl ...",	string, null	
"selfSignature": "c2lnbmF0dXJlIG9mIHRo ...",	string, null	
"signature": "c2lnbmF0dXJlIG9mIHRo ..."	string, null	
}		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null

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`
default: null

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 null if not supported or required. Property value constraints: `format: base64` default: null

signature
Specifies the RSA signature of the data item exported.
This should be null if not supported or required.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`
default: null

Event Messages

None

11.2.10 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 2.0)	Type	Required
{		
"getCertificate": "enckey"	string	✓
}		

Properties

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 2.0)	Type	Required
{		
"errorCode": "accessDenied",	string, null	
"certificate": "Y2VydGlmaWNhdGUgREVS ..."	string, null	
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

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
```

default: null

Event Messages

None

11.2.11 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 2.0)	Type	Required
{		
"replaceCertificate": "UEtDUyAjNyBkYXRh"	string	✓
}		

Properties

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 2.0)	Type	Required
{		
"errorCode": "accessDenied",	string, null	
"newCertificateData": "UEtDUyAjNyB0aHVtYiBw ..."	string, null	
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

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
```

default: null

Event Messages

None

11.2.12 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 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "accessDenied",`	string, null	
`"randomItem": "Tm9uY2U="`	string, null	
`}`		

Properties

errorCode

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

- `accessDenied` - The encryption module is either not initialized or not ready for any vendor specific reason.

default: null

randomItem

The randomly generated number created by the device.

This value is null if the device does not support random number generation for data authentication.

Property value constraints:

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

default: null

Event Messages

None

11.2.13 KeyManagement.GenerateKCV

This command returns the Key Check Value (KCV) for the specified key.

Command Message

Payload (version 2.0)	Type	Required
{		
"key": "Key01",	string	✓
"keyCheckMode": "self"	string	✓
}		

Properties

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 2.0)	Type	Required
{		
"errorCode": "keyNotFound",	string, null	
"kcv": "a2N2"	string, null	
}		

Properties

errorCode
Specifies the error code if applicable, otherwise null. 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.

default: null

kcv
Contains KCV data that can be used for verification of the key. If the command fails, this will be null.
Property value constraints:

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

default: null

Event Messages

None

11.2.14 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 2.0)	Type	Required
{		
"loadOption": "newHost",	string	✓
"signer": "certHost",	string	✓
"certificateData": "Y2VydGlmaWNhdGUgaW4g ..."	string	✓
}		

Properties

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 *newHost*, 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 2.0)	Type	Required
{		
"errorCode": "accessDenied",	string, null	
"rsaKeyCheckMode": "none",	string	
"rsaData": "UEtDUyAjNyBkYXRh"	string, null	
}		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null
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. default: "none"

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

Event Messages

None

11.2.15 KeyManagement.StartAuthenticate

For commands which may require authentication data, this command retrieves the data to be signed. If this command returns data to be signed the signed data must be included in the *authenticate* property of the command.

If authentication data is required but not provided, the command will complete with completionCode authorizationRequired.

Command Message

Payload (version 2.0)	Type	Required
{		
"command": {	object	✓
"deleteKey": {	object, null	
"key": "Key01"	string	✓
},		
"initialization": {	object, null	
}		
}		
}		

Properties

command

The command and associated command specific input properties which for which data to sign is requested. This must be one of:

- `deleteKey` - The KeyManagement.DeleteKey command.
- `initialization` - KeyManagement.Initialization command.

Property value constraints:

```
minProperties: 1
maxProperties: 1
```

command/deleteKey

See KeyManagement.DeleteKey description.

default: null

command/deleteKey/key

The name of key being deleted.

command/initialization

See KeyManagement.Initialization description.

default: null

Completion Message

Payload (version 2.0)	Type	Required
{		
"dataToSign": "QXV0aGVudGljYXRpb24g ...",	string, null	
"signers": {	object, null	
"certHost": false,	boolean	
"sigHost": false,	boolean	
"ca": false,	boolean	
"hl": false,	boolean	

Payload (version 2.0)	Type	Required
"cbcmac": false,	boolean	
"cmac": false,	boolean	
"certHostTr34": false,	boolean	
"caTr34": false,	boolean	
"hlTr34": false,	boolean	
"reserved1": false,	boolean	
"reserved2": See signers/reserved1,	boolean	
"reserved3": See signers/reserved1	boolean	
}		
}		

Properties

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 null and the command result is success.

Property value constraints:

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

default: null

signers

Specifies the methods which may be used to generate authentication data. If *dataToSign* is not null, at least one method must be true.

default: null

signers/certHost

The current host can be used to generate authentication data, using the RSA certificate-based scheme.

default: false

signers/sigHost

The current host can be used to generate authentication data, using the RSA signature-based scheme.

default: false

signers/ca

The Certificate Authority (CA) can be used to generate authentication data.

default: false

signers/hl

The Higher Level (HL) Authority can be used to generate authentication data.

default: false

signers/cbcmac

A CBC MAC key can be used to generate authentication data.

default: false

signers/cmac

A CMAC key can be used to generate authentication data.

default: false

signers/certHostTr34

The current host can be used to generate authentication data compliant with TR-34.

default: false

Properties
signers/caTr34 The Certificate Authority (CA) can be used to generate the authentication data compliant with TR-34. default: false
signers/hlTr34 The Higher Level (HL) Authority can be used to generate authentication data compliant with TR-34. default: false
signers/reserved1 The authentication data is generated using a vendor specific generation method. default: false

Event Messages

None

11.2.16 KeyManagement.ImportKeyToken

This command is used to load a DES (56, 112, 168) or AES (128, 192, 256) key included in a key transport token. The Key Transport Key should be destroyed if the entire process is not completed. In addition, a new Key Transport Key should be generated each time this protocol is executed. This command ends the Key Exchange process.

Command Message

Payload (version 2.0)	Type	Required
{		
"keyToken": "UGluYmxvY2sgZGF0YQ==",	string	✓
"key": "Key01",	string	✓
"keyUsage": "P0",	string, null	
"loadOption": "noRandom"	string	
}		

Properties

keyToken

Pointer to a binary encoded PKCS #7 represented in DER encoded ASN.1 notation. This allows the Host to verify that key was imported correctly and to the correct device. The message has an outer Signed-data content type with the SignerInfo encryptedDigest field containing the HOST′s signature. The inner content is an Enveloped-data content type. The device identifier is included as the issuerAndSerialNumber within the RecipientInfo.

Property value constraints:

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

key

Specifies the name of the key to be stored.

Properties

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]$
```

default: null

Properties
loadOption Specifies the method to use to load the key token as one of the following values: • `noRandom` - Import a key without generating a using a random number. • `random` - Import a key by generating and using a random number. This option is used for Remote Key Exchange • `noRandomCrl` - Import a key with a Certificate Revocation List included in the token. A random number is not generated nor used. This option is used for the One-Pass Protocol described in X9 TR34-2019 [Ref. keymanagement-9] • `randomCrl` - Import a key with a Certificate Revocation List included in the token. A random number is generated and used. This option is used for the Two-Pass Protocol described in X9 TR34-2019 [Ref. keymanagement-9] If *random* or *randomCrl*, the random number is included as an authenticated attribute within SignerInfo SignedAttributes. If *noRandom* or *noRandomCrl*, a timestamp is included as an authenticated attribute within SignerInfo SignedAttributes. If *noRandomCrl* or *randomCrl*, *keyUsage* must be null as the key usage is embedded in the *keyToken*. default: "noRandom"

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "accessDenied",`	string, null	
`"keyLength": 0,`	integer	
`"keyAcceptAlgorithm": "sha1",`	string, null	
`"keyAcceptData": "UGluYmxvY2sgZGF0YQ==",`	string, null	
`"keyCheckMode": "kcvSelf",`	string, null	
`"keyCheckValue": "MDEwMjAz"`	string, null	
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. The following values are possible: • `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. • `invalidKeyLength` - The length of the Key Transport Key is not valid. • `noKeyRam` - There is no space left in the key RAM for a key of the specified type. • `formatInvalid` - The format of the message or key block is invalid. • `contentInvalid` - The content of the message or key block is invalid. • `useViolation` - The specified use is not supported, or if a key with the same name has already been loaded, the specified use conflicts with the use of the key previously loaded. • `randomInvalid` - The encrypted random number in the input data does not match the one previously provided by the PIN device. Only applies to CRKL load options that use a random number. • `signatureInvalid` - The signature in the input data is invalid. • `invalidCertState` - A Host certificate has not been previously loaded. default: null

Properties
keyLength Specifies the length, in bits, of the key. Zero if the key length is unknown. Property value constraints: `minimum: 0` default: 0
keyAcceptAlgorithm Defines the algorithm used to generate the signature contained in the message *keyAcceptData* sent to the host. The following values are possible: • `sha1` - *keyAcceptData* contains a SHA-1 digest of concatenated data using the device signing key. • `sha256` - *keyAcceptData* contains a SHA-256 digest of concatenated data using the device signing key. default: null

keyAcceptData
If *loadOption* is *random* or *randomCrl*, this data is a binary encoded PKCS #7, represented in DER encoded ASN.1 notation. The message has an outer Signed-data content type with the SignerInfo encryptedDigest field containing the ATM′s signature. The random numbers are included as authenticatedAttributes within the SignerInfo. The inner content is a data content type, which contains the HOST identifier as an issuerAndSerialNumber sequence.
If *keyAcceptAlgorithm* is null, then this will also be null.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]*={0,2}$`
`format: base64`
default: null

keyCheckMode Specifies the mode that is used to create the key check value. The following values are possible: • `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. default: null

keyCheckValue
Contains the key verification code data that can be used for verification of the loaded key. This will be null if the device does not have that capability.
If *keyCheckMode* is null, then this will also be null.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]*={0,2}$`
`format: base64`
default: null

Event Messages

None

11.2.17 KeyManagement.ImportEmvPublicKey

The Certification Authority and the Chip Card RSA public keys needed for EMV are loaded or deleted in/from the encryption module. This command is similar to the KeyManagement.ImportKey, but it is specifically designed to address the key formats and security features defined by EMV. Mainly the extensive use of "signed certificate" or "EMV certificate" (which is a compromise between signature and a pure certificate) to provide the public key is taken in account. The Service is responsible for all EMV public key import validation. Once loaded, the Service is not responsible for key/certificate expiry, this is an application responsibility.

Command Message

Payload (version 2.0)	Type	Required
{		
"key": "Key01",	string	✓
"keyUsage": "E0",	string	✓
"importScheme": "plainCA",	string	✓
"value": "string",	string	✓
"verifyKey": "Key01"	string, null	
}		

Properties

key
Specifies the name of key being loaded.

keyUsage
Specifies the type of access for which the *key* can be used. The following values are possible:

- `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.
- `00 - 99` - These numeric values are reserved for proprietary use.

Property value constraints:

```
pattern: ^E[0-7]$|^[0-9][0-9]$
```

importScheme
Defines the import scheme used. The following values are possible:

- `plainCA` - This scheme is used by VISA. A plain text CA public key is imported with no verification.

The two parts of the key (modulus and exponent) are passed in clear mode as a DER encoded PKCS#1 public key. The key is loaded directly in the security module.

- `checksumCA` - This scheme is used by VISA. A plain text CA public key is imported using the EMV 2000

Book II verification algorithm and it is verified before being loaded in the security module.

- `epiCA` - This scheme is used by MasterCard Europe. A CA public key is imported using the self-signed scheme.
- `issuer` - An Issuer public key is imported as defined in EMV 2000 Book II.
- `icc` - An ICC public key is imported as defined in EMV 2000 Book II.
- `iccPIN` - An ICC PIN public key is imported as defined in EMV 2000 Book II.
- `pkcsV1_5_CA` - A CA public key is imported and verified using a signature generated with a private key for which the public key is already loaded.

Properties

value

Contains all the necessary data to complete the import using the scheme specified within *importScheme*.

If *importScheme* is *plainCA* then *value* contains a DER encoded PKCS#1 public key. No verification is possible. *verifyKey* is ignored.

If *importScheme* is *checksumCA* then *value* contains table 23 data, as specified in EMV 2000 Book 2 (See [Ref. keymanagement-3]). The plain text key is verified as defined within EMV 2000 Book 2, page 73. *verifyKey* is ignored (See [Ref. keymanagement-3]).

If *importScheme* is WFS_PIN_EMV_IMPORT_EPI_CA then *value* contains the concatenation of tables 4 and 13, as specified in [Ref. keymanagement-4], Europay International, EPI CA Module Technical – Interface specification Version 1.4. These tables are also described in the EMV Support Appendix. The self-signed public key is verified as defined by the reference document. *sigKey* is ignored.

If *importScheme* is *issuer* then *value* contains the EMV public key certificate. Within the following descriptions tags are documented to indicate the source of the data, but they are not sent down to the service. The data consists of the concatenation of: the key exponent length (1 byte), the key exponent value (variable length – EMV Tag value: '9F32'), the EMV certificate length (1 byte), the EMV certificate value (variable length – EMV Tag value: '90'), the remainder length (1 byte). The remainder value (variable length – EMV Tag value: '92'), the PAN length (1 byte) and the PAN value (variable length – EMV Tag value: '5A'). The service will compare the leftmost three to eight hex digits (where each byte consists of two hex digits) of the PAN to the Issuer Identification Number retrieved from the certificate. For more explanations, the reader can refer to EMVCo, Book2 – Security & Key Management Version 4.0, Table 4 (See [Ref. keymanagement-3]). *verifyKey* defines the previously loaded key used to verify the signature.

If *importScheme* is *icc* then *value* contains the EMV public key certificate. Within the following descriptions tags are documented to indicate the source of the data, but they are not sent down to the service. The data consists of the concatenation of: the key exponent length (1 byte), the key exponent value (variable length– EMV Tag value: '9F47'), the EMV certificate length (1 byte), the EMV certificate value (variable length – EMV Tag value:'9F46'), the remainder length (1 byte), the remainder value (variable length – EMV Tag value: '9F48'), the SDA length (1 byte), the SDA value (variable length), the PAN length (1 byte) and the PAN value (variable length – EMV Tag value: '5A'). The service will compare the PAN to the PAN retrieved from the certificate. For more explanations, the reader can refer to EMVCo, Book2 – Security & Key Management Version 4.0, Table 9 (See [Ref. keymanagement-3]). *verifyKey* defines the previously loaded key used to verify the signature.

If *importScheme* is *iccPIN* then *value* contains the EMV public key certificate. Within the following descriptions tags are documented to indicate the source of the data, but they are not sent down to the service. The data consists of the concatenation of: the key exponent length (1 byte), the key exponent value (variable length – EMV Tag value: '9F2E'), the EMV certificate length (1 byte), the EMV certificate value (variable length – EMV Tag value:'9F2D'), the remainder length (1 byte), the remainder value (variable length – EMV Tag value: '9F2F'), the SDA length (1 byte), the SDA value (variable length), the PAN length (1 byte) and the PAN value (variable length – EMV Tag value: '5A'). The service will compare the PAN to the PAN retrieved from the certificate. For more explanations, the reader can refer to EMVCo, Book2 – Security & Key Management Version 4.0, Table 9 (See [Ref. keymanagement-3]). *verifyKey* defines the previously loaded key used to verify the signature.

If *importScheme* is *pkcsV1_5_CA* then *value* contains the CA public key signed with the previously loaded public key specified in *verifyKey*. *value* consists of the concatenation of EMV 2000 Book II Table 23 + 8 byte random number + Signature (See [Ref. keymanagement-3]). The 8-byte random number is not used for validation; it is used to ensure the signature is unique. The Signature consists of all the bytes in the *value* buffer after table 23 and the 8-byte random number.

Property value constraints:

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

verifyKey

Specifies the name of the previously loaded key used to verify the signature, as detailed in the descriptions above.

default: null

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "accessDenied",	string, null	
"expiryDate": "0123"	string, null	
}		

Properties

errorCode

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

- `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.
- `noKeyRam` - There is no space left in the key RAM for a key of the specified type.
- `emvVerifyFailed` - The verification of the imported key failed and the key was discarded.
- `keyNotFound` - The specified *key* was not found.
- `useViolation` - The specified *keyUsage* is not supported by this key.

default: null

expiryDate

Contains the expiry date of the certificate in the following format MMYY. If null, the certificate does not have an expiry date.

default: null

Event Messages

None

11.3 Event Messages

11.3.1 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 2.0)	Type	Required
{		
"key": "Key01",	string	✓
"ksn": "S1NORGF0YQ=="	string	✓
}		

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
```

11.4 Unsolicited Messages

11.4.1 KeyManagement.InitializedEvent

This is generated when a KeyManagement.Initialization command completed successfully.

Unsolicited Message

Payload (version 2.0)
`This message does not define any properties.`

11.4.2 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 2.0)	Type	Required
{		
"keyName": "Key02",	string	✓
"errorCode": "keyNotFound"	string	✓
}		

Properties

keyName
Specifies the name of the key that caused the error.

errorCode
Specifies the type of illegal key access that occurred 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.

11.4.3 KeyManagement.CertificateChangeEvent

This event indicates that the certificate module state has changed from Primary to Secondary.

Unsolicited Message

Payload (version 2.0)	Type	Required
{		
"certificateChange": "secondary"	string	
}		

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.

default: "secondary"

12. Crypto Interface

This chapter defines the Crypto interface functionality and messages.

12.1 General Information

12.1.1 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

12.2 Command Messages

12.2.1 Crypto.GenerateRandom

This command is used to generate a random number.

Command Message

Payload (version 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "accessDenied",`	string, null	
`"randomNumber": "VGhlIGdlbmVyYXRlZCBy ..."`	string, null	
`}`		

Properties

errorCode

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

- `accessDenied` - The encryption module is either not initialized or not ready for any vendor specific reason.

default: null

randomNumber

The random number. If the command fails, this will be null.

Property value constraints:

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

default: null

Event Messages

None

12.2.2 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 2.0)	Type	Required
{		
"key": "Key001",	string	✓
"storedKey": "StoredIVKey",	string, null	
"iv": {	object, null	
"key": "KeyToDecrypt",	string, null	
"value": "VGhlIGluaXRpYWxpemF0 ..."	string	✓
},		
"padding": 255,	integer	
"modeOfUse": "E",	string, null	
"cryptoMethod": "ecb",	string	✓
"data": "U2FtcGxlIERhdGE="	string	✓
}		

Properties

key
Specifies the name of the encryption key. The key usage must one of the supported *cryptoAttributes*.

storedKey
This specifies the name of a key (usage 'I0') used as the Initialization Vector (IV). This property is null if not required.
default: null

iv
Specifies the Initialization Vector. This property is null if *storedKey* is used.
default: null

iv/key
The name of a key used to decrypt the *value*. This specifies the name of a key (usage 'K0') used to decrypt the *value*. This is only used when the *key* usage is 'D0' and *cryptoMethod* is either CBC or CFB. if this property is null, *value* is used as the Initialization Vector.
default: null

iv/value
The plaintext or encrypted IV for use with the CBC or CFB encryption methods.
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`
default: 0

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 null. Property value constraints: `pattern: ^[DE]$` default: null
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 2.0)	Type	Required
`{`		
`"errorCode": "accessDenied",`	string, null	
`"data": "U2FtcGxlIERhdGE="`	string, null	
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null

data
The encrypted or decrypted data. If the command fails, this will be null.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`
default: null

Event Messages

- KeyManagement.DUKPTKSNEvent

12.2.3 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 2.0)	Type	Required
{		
"key": "Key001",	string	✓
"data": "VGhlIEJhc2U2NCBlbmNv ...",	string	✓
"storedKey": "StoredIVKey",	string, null	
"iv": {	object, null	
"key": "KeyToDecrypt",	string, null	
"value": "VGhlIGluaXRpYWxpemF0 ..."	string	✓
},		
"padding": 255,	integer	
"compression": "@",	string, null	
"cryptoMethod": "rsassaPkcs1V15",	string, null	
"hashAlgorithm": "sha1",	string, null	
"authenticationDatalength": 4	integer	
}		

Properties

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
```

storedKey
This specifies the name of a key (usage 'I0') used as the Initialization Vector (IV). This property is null if not required.
default: null

iv
Specifies the Initialization Vector. This property is null if *storedKey* is used.
default: null

iv/key
The name of a key used to decrypt the *value*. This specifies the name of a key (usage 'K0') used to decrypt the *value*. This is only used when the *key* usage is 'D0' and *cryptoMethod* is either CBC or CFB. if this property is null, *value* is used as the Initialization Vector.
default: null

Properties

iv/value
The plaintext or encrypted IV for use with the CBC or CFB encryption methods.
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`
default: 0

compression Specifies whether the data is to be compressed (blanks removed) before building the MAC. If this property is null, the compression is not applied. Otherwise this property value is the blank character (e.g. ' ' in ASCII or '@' in EBCDIC). Property value constraints: `pattern: ^[ @]$` default: null
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 null. default: null
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. • `na` - Not applicable. [Ref. crypto-1] and FIPS 180-2 [Ref. crypto-2]. If the *key* usage is a MAC usage (e.g. 'M0') this property will be ignored. default: null

authenticationDatalength
The required authentication data length.
If the *key* usage is an asymmetric key pair signature usage (e.g. 'S0') this property will be ignored.
Property value constraints:
`minimum: 4`
`maximum: 8`
default: 4

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "accessDenied",`	string, null	
`"authenticationData": "VGhlIG1hYyB2YWx1ZSBv ..."`	string, null	
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null

authenticationData
The generated authentication data. If the command fails, this will be null.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`
default: null

Event Messages

- KeyManagement.DUKPTKSNEvent

12.2.4 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 2.0)	Type	Required
{		
"key": "Key001",	string	✓
"data": "R2VuZXJhdGUgYSBNQUMg ...",	string	✓
"verifyData": "RGF0YSB0byBiZSB2ZXJp ...",	string	✓
"storedKey": "StoredIVKey",	string, null	
"iv": {	object, null	
"key": "KeyToDecrypt",	string, null	
"value": "VGhlIGluaXRpYWxpemF0 ..."	string	✓
},		
"padding": 255,	integer	
"compression": "@",	string, null	
"cryptoMethod": "rsassaPkcs1V15",	string, null	
"hashAlgorithm": "sha1"	string, null	
}		

Properties

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
```

storedKey
This specifies the name of a key (usage 'I0') used as the Initialization Vector (IV). This property is null if not required.
default: null

Properties
iv Specifies the Initialization Vector. This property is null if *storedKey* is used. default: null
iv/key The name of a key used to decrypt the *value*. This specifies the name of a key (usage 'K0') used to decrypt the *value*. This is only used when the *key* usage is 'D0' and *cryptoMethod* is either CBC or CFB. if this property is null, *value* is used as the Initialization Vector. default: null

iv/value
The plaintext or encrypted IV for use with the CBC or CFB encryption methods.
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`
default: 0

compression Specifies whether the data is to be compressed (blanks removed) before building the MAC. If this property is null, the compression is not applied. Otherwise this property value is the blank character (e.g. ' ' in ASCII or '@' in EBCDIC). Property value constraints: `pattern: ^[ @]$` default: null
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. • `na` - Not applicable. If the *key* usage is a MAC usage (e.g. 'M0') this property should be null. default: null
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. • `na` - Not applicable. [Ref. crypto-1] and FIPS 180-2 [Ref. crypto-2]. If the *key* usage is a MAC usage (e.g. 'M0') this property will be ignored. default: null

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "accessDenied"`	string, null	

Payload (version 2.0)	Type	Required
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

Event Messages

- KeyManagement.DUKPTKSNEvent

12.2.5 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 2.0)	Type	Required
{		
"hashAlgorithm": "sha1",	string	✓
"data": "U2FtcGxlIERhdGE="	string	✓
}		

Properties

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 2.0)	Type	Required
{		
"errorCode": "accessDenied",	string, null	
"digest": "OTNjYzE2Y2FkNzYwMTY3 ..."	string, null	
}		

Properties

errorCode

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

- `accessDenied` - The encryption module is either not initialized or not ready for any vendor specific reason.

default: null

digest

Contains the generated digest. If the command fails, this will be null.

Property value constraints:

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

default: null

Event Messages

None

13. Keyboard Interface

This chapter defines the Keyboard interface functionality and messages.

This section describes the general interface for the following functions:

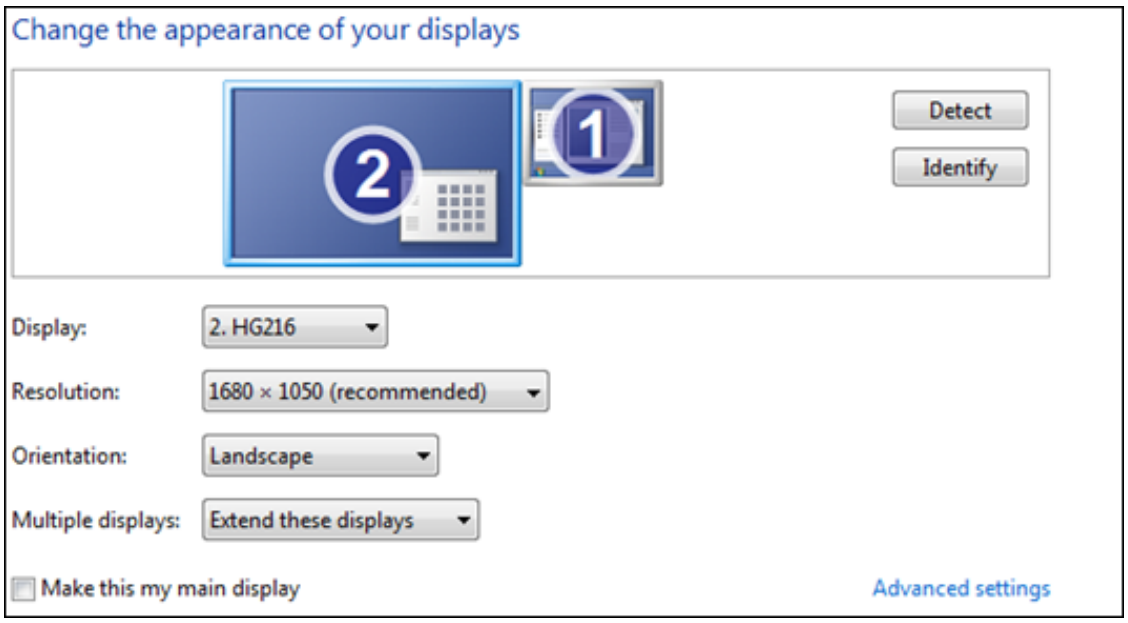
- Entering Personal Identification Numbers (PINs)
- Clear text data handling
- Function key handling
  If the device has local display capability, display handling should be handled using the TextTerminal interface. The adoption of this specification does not imply the adoption of a specific security standard.
- Only numeric PIN pads are handled in this specification.

13.1 General Information

13.1.1 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:

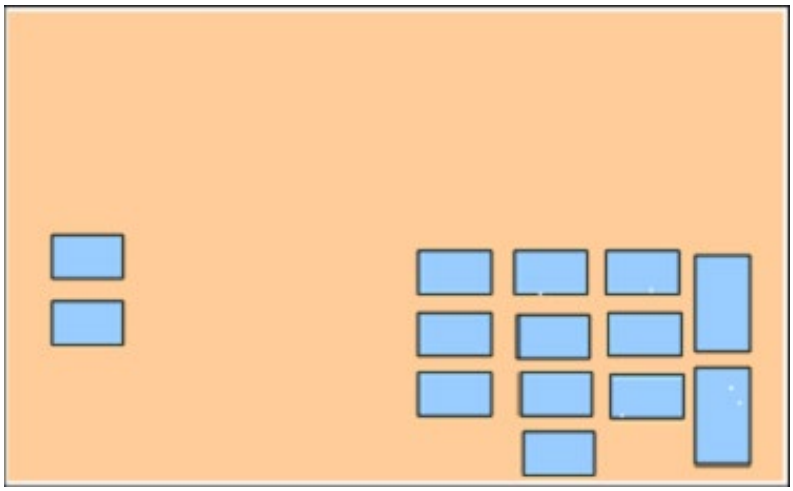
- Keyboard.PinEntry
- Keyboard.DataEntry
- Keyboard.SecureKeyEntry

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.

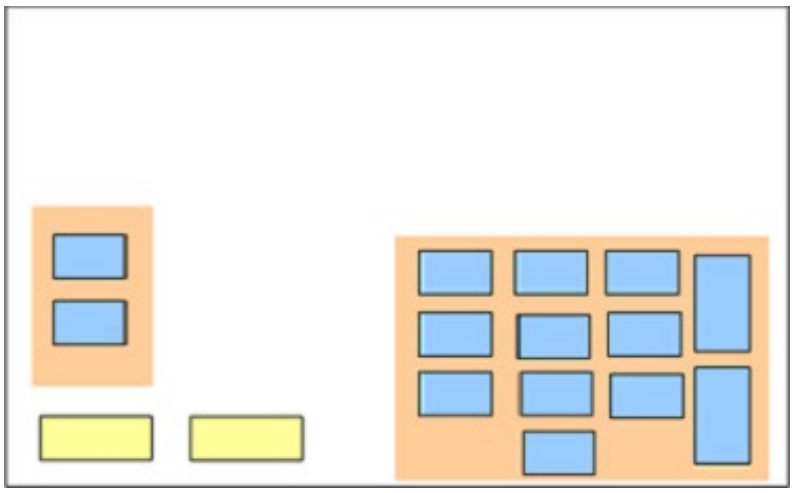
Layouts defined with the Keyboard.DefineLayout command are persistent.

Example 1 – this screen only uses Data mode – the entire screen is a Touch Frame. Mouse mode is not used.

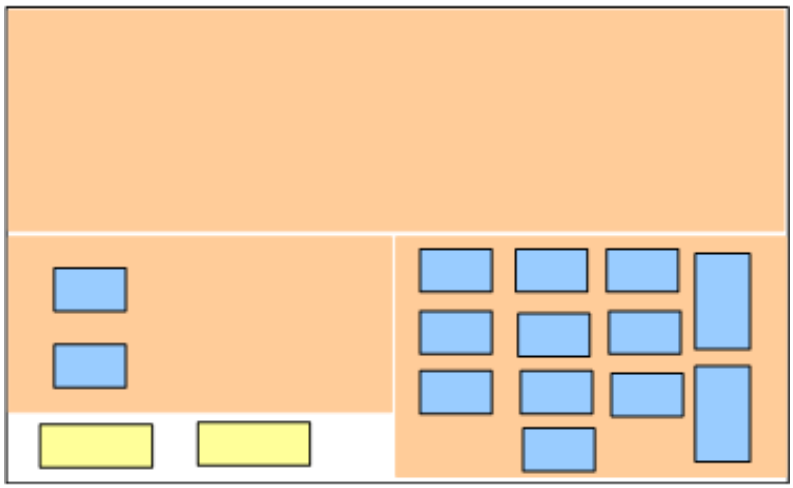


Example 2 – this shows a monitor with two Touch Frames and 14 Touch Keys. The space within the Touch Frames not defined by a Touch Key are inactive (do not respond to touch). All areas outside a Touch Frame operate in

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.



13.1.2 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*

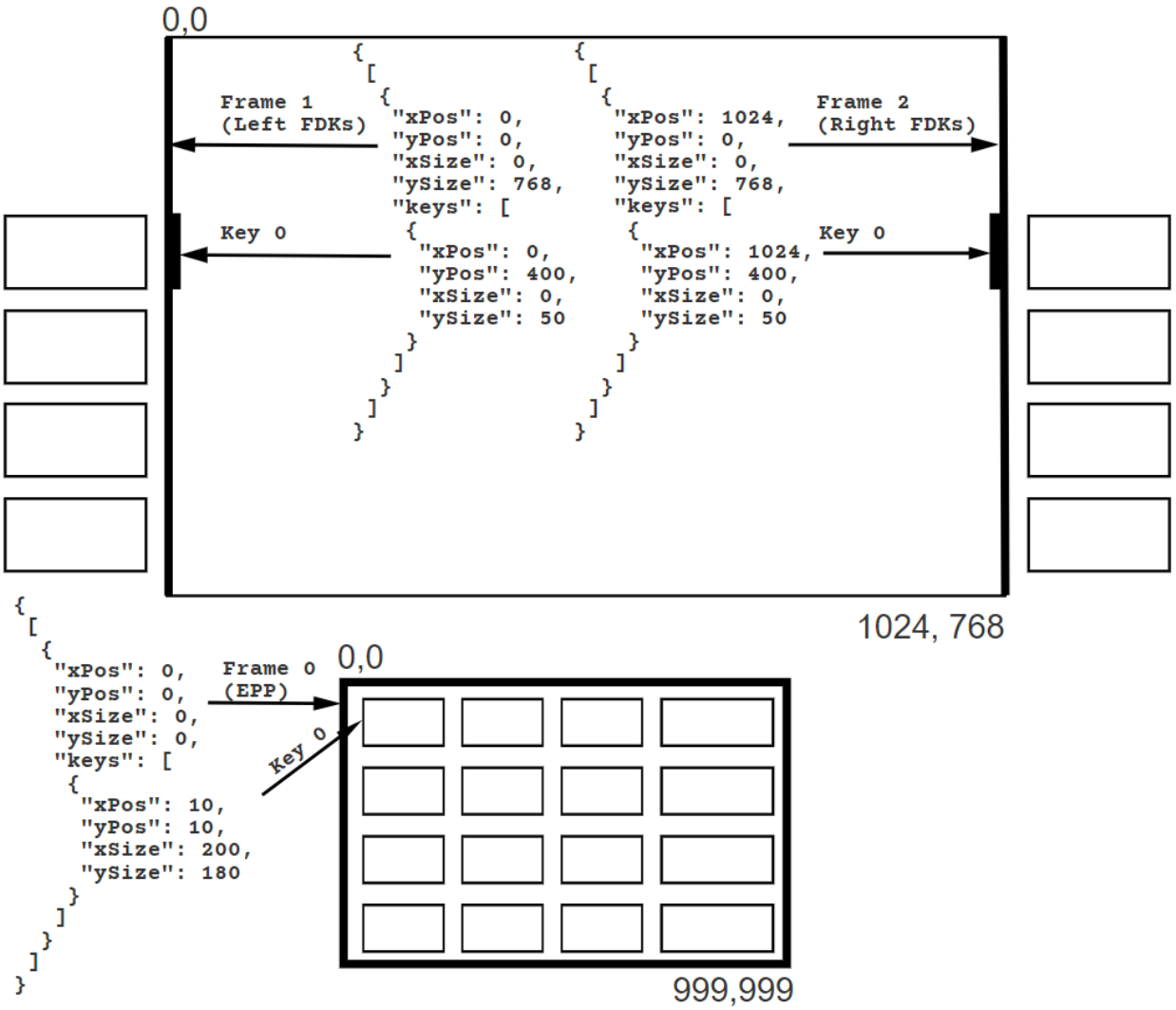
[1]: 1 is the smallest possible size and 1000 is the full width of the frame

[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.



13.2 Command Messages

13.2.1 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 2.0)	Type	Required
{		
"entryMode": "data"	string, null	
}		

Properties

entryMode

Specifies entry mode to be returned. If this property is null, all layouts for the Keyboard.DataEntry, Keyboard.PinEntry and Keyboard.SecureKeyEntry command are 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.

default: null

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "modeNotSupported",	string, null	
"layout": {	object, null	
"data": [{	array (object), null	
"xPos": 0,	integer	✓
"yPos": 0,	integer	✓
"xSize": 0,	integer	✓
"ySize": 0,	integer	✓
"float": {	object, null	
"x": false,	boolean	
"y": false	boolean	
},		
"keys": [{	array (object)	✓
"key": "one",	string	✓

Payload (version 2.0)	Type	Required
"xPos": 0,	integer	✓
"yPos": 0,	integer	✓
"xSize": 1,	integer	✓
"ySize": 1,	integer	✓
"shiftKey": "a"	string	✓
}]		
}],		
"pin": See layout/data properties	array (object), null	
"secure": See layout/data properties	array (object), null	
}		
}		

Properties

errorCode

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

- `modeNotSupported` - The specified entry mode is not supported.

default: null

layout

Return supported layouts specified by the *entryMode* property.

default: null

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.

default: null

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
```

Properties
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. If etsCaps is null or float is null, this property is null. If this property is null, the device cannot randomly shift the layout in both horizontal and vertical direction. default: null
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. Property value constraints: `minItems: 1`

Properties

layout/data/keys/key

Specifies the Function Key associated with the physical area in non-shifted mode.

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/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/shiftKey Specifies the Function Key associated with the physical key in shifted mode. 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. default: null
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. default: null

Event Messages

None

13.2.2 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 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 2.0)	Type	Required
{		
"minLen": 0,	integer	✓
"maxLen": 0,	integer	
"autoEnd": false,	boolean	
"echo": "X",	string	
"activeKeys": {	object	✓
"*one*": {	object	
"terminate": false	boolean	
},		
"*backspace*": See *activeKeys/one* properties	object	

Payload (version 2.0)	Type	Required
} }		

Properties

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
```

default: 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. This property will be ignored by the service if the device doesn't have a local display.

Property value constraints:

```
minLength: 1
maxLength: 1
```

default: "X"

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.

Properties
activeKeys/one (example name) An active key. 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 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 2.0)	Type	Required
{		
"errorCode": "keyInvalid",	string, null	
"digits": 0,	integer	
"completion": "auto"	string, null	
}		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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). default: null
digits Specifies the number of PIN digits entered. Property value constraints: `minimum: 0` default: 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. 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). default: null

Event Messages

- Keyboard.KeyEvent
- Keyboard.EnterDataEvent
- Keyboard.LayoutEvent

13.2.3 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 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 2.0)	Type	Required
{		
"maxLen": 0,	integer	
"autoEnd": false,	boolean	
"activeKeys": {	object	✓
"*one*": {	object	
"terminate": false	boolean	
},		
"*backspace*": See *activeKeys/one* properties	object	
}		
}		

Properties
maxLen Specifies the maximum number of digits which can be returned to the application in the output parameter. Property value constraints: `minimum: 0` default: 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.
activeKeys/one (example name) An active key. 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 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 2.0)	Type	Required
{		

Payload (version 2.0)	Type	Required
"errorCode": "keyInvalid",	string, null	
"keys": 0,	integer	
"pinKeys": [{	array (object), null	
"completion": "auto",	string, null	
"digit": "five"	string	✓
}],		
"completion": See pinKeys/completion	string, null	
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

keys

Number of keys entered by the user

Property value constraints:

```
minimum: 0
```

default: 0

pinKeys

Array contains the keys entered by the user

default: null

pinKeys/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. 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).

default: null

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 null 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]*)$
```

Event Messages

- Keyboard.KeyEvent
- Keyboard.EnterDataEvent
- Keyboard.LayoutEvent

13.2.4 Keyboard.Reset

Sends a service reset to the Service Provider.

Command Message

Payload (version 2.0)
This message does not define any properties.

Completion Message

Payload (version 2.0)
This message does not define any properties.

Event Messages

None

13.2.5 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 2.0)	Type	Required
{		
"keyLen": 0,	integer	✓
"autoEnd": false,	boolean	
"activeKeys": {	object	✓
"*one*": {	object	
"terminate": false	boolean	
},		
"*backspace*": See *activeKeys/one* properties	object	
},		
"verificationType": "self",	string	✓
"cryptoMethod": "des"	string, null	

Payload (version 2.0)	Type	Required
}		

Properties

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.

activeKeys/one (example name)

An active key.

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

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

Properties
verificationType Specifies the type of verification to be done on the entered key. The following values are possible: • `self` - The key check value is created by an encryption of the key with itself. For a double-length or triple-length key the KCV is generated using 3DES encryption using the first 8 bytes of the key as the source data for the encryption. • `zero` - The key check value is created by encrypting a zero value with the key.
cryptoMethod Specifies the cryptographic method to be used for the verification. If this property is null, *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 default: null

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "accessDenied",`	string, null	
`"digits": 0,`	integer	
`"completion": "auto",`	string, null	
`"kcv": "S2V5IENoZWNrIFZhbHVl"`	string	
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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). default: null
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` default: 0

Properties
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. 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). default: null

kcv
Contains the key check value data that can be used for verification of the entered key formatted in Base64. This value is null 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`
default: ""

Event Messages

- Keyboard.KeyEvent
- Keyboard.EnterDataEvent
- Keyboard.LayoutEvent

13.2.6 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 2.0)	Type	Required
{		
"mode": {	object	✓
"active": false,	boolean	
"inactive": false	boolean	
}		
}		

Properties
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 2.0)
This message does not define any properties.

Event Messages

None

13.2.7 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 2.0)	Type	Required
{		
"layout": {	object	✓
"data": [{	array (object), null	
"xPos": 0,	integer	✓
"yPos": 0,	integer	✓
"xSize": 0,	integer	✓
"ySize": 0,	integer	✓
"float": {	object, null	
"x": false,	boolean	
"y": false	boolean	
},		
"keys": [{	array (object)	✓
"key": "one",	string	✓
"xPos": 0,	integer	✓
"yPos": 0,	integer	✓
"xSize": 1,	integer	✓
"ySize": 1,	integer	✓
"shiftKey": "a"	string	✓
}]		
}],		
"pin": See layout/data properties	array (object), null	

Payload (version 2.0)	Type	Required
"secure": See layout/data properties	array (object), null	
}		
}		

Properties

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.
default: null

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
```

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. If etsCaps is null or float is null, this property is null. If this property is null, the device cannot randomly shift the layout in both horizontal and vertical direction.
default: null

Properties

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.

Property value constraints:

```
minItems: 1
```

layout/data/keys/key

Specifies the Function Key associated with the physical area in non-shifted mode.

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/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
```

Properties

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`

layout/data/keys/shiftKey Specifies the Function Key associated with the physical key in shifted mode. 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. default: null
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. default: null

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "modeNotSupported"`	string, null	
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null

Event Messages

None

13.3 Event Messages

13.3.1 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 2.0)	Type	Required
{		
"completion": "auto",	string, null	
"digit": "five"	string	✓
}		

Properties

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. 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).

default: null

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 null 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]*)$
```

13.3.2 Keyboard.EnterDataEvent

This mandatory event notifies the application when the device is ready for the user to start entering data.

Event Message

Payload (version 2.0)
This message does not define any properties.

13.3.3 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 2.0)	Type	Required
{		
"data": [{	array (object), null	
"xPos": 0,	integer	✓
"yPos": 0,	integer	✓
"xSize": 0,	integer	✓
"ySize": 0,	integer	✓
"float": {	object, null	
"x": false,	boolean	
"y": false	boolean	
},		
"keys": [{	array (object)	✓
"key": "one",	string	✓
"xPos": 0,	integer	✓
"yPos": 0,	integer	✓
"xSize": 1,	integer	✓
"ySize": 1,	integer	✓
"shiftKey": "a"	string	✓
}]		
}],		
"pin": See data properties	array (object), null	
"secure": See data properties	array (object), null	
}		

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.

default: null

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
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. If etsCaps is null or float is null, this property is null. If this property is null, the device cannot randomly shift the layout in both horizontal and vertical direction. default: null
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. Property value constraints: `minItems: 1`

Properties

data/keys/key

Specifies the Function Key associated with the physical area in non-shifted mode.

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]*)$
```

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
```

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
```

Properties
data/keys/shiftKey Specifies the Function Key associated with the physical key in shifted mode. 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. default: null
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. default: null

14. 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

14.1 General Information

14.1.1 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

14.2 Command Messages

14.2.1 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 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"manufacturerIdentifier": "Manufacturer ID",`	string, null	
`"modelIdentifier": "Model ID",`	string, null	
`"hardwareIdentifier": "Hardware ID",`	string, null	
`"firmwareIdentifier": "Firmware ID",`	string, null	
`"applicationIdentifier": "Application ID"`	string, null	
`}`		

Properties

manufacturerIdentifier
Returns the manufacturer identifier of the PIN device. This value is null 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.
default: null

modelIdentifier
Returns the model identifier of the PIN device. This value is null if the model identifier is not available.
default: null

hardwareIdentifier
Returns the hardware identifier of the PIN device. This value is null if the hardware identifier is not available.
default: null

firmwareIdentifier
Returns the firmware identifier of the PIN device. This value is null if the firmware identifier is not available.
default: null

applicationIdentifier
Returns the application identifier of the PIN device. This value is null if the application identifier is not available.
default: null

Event Messages

None

14.2.2 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 2.0)	Type	Required
{		
"validationData": "0812746533758375",	string	✓
"offset": "0000000000000000",	string, null	
"padding": "00",	string	
"maxPIN": 0,	integer	✓
"valDigits": 0,	integer	✓
"noLeadingZero": false,	boolean	✓
"key": "Key01",	string	✓
"keyEncKey": "Key02",	string, null	
"decTable": "3183042102277795"	string	✓
}		

Properties

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 null 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}$
```

default: null

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 value is null, *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. default: null
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 2.0)	Type	Required
`{`		
`"errorCode": "keyNotFound",`	string, null	
`"result": false`	boolean	
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null
result Specifies whether the PIN is correct or not. default: false

Event Messages

None

14.2.3 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 2.0)	Type	Required
{		
"pan": "01234567890123456789123",	string	✓
"pvv": "0286",	string	✓
"key": "Key01",	string	✓
"keyEncKey": "UGluYmxvY2sgZGF0YQ=="	string, null	
}		

Properties

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

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "keyNotFound",	string, null	
"result": false	boolean	
}		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null
result Specifies whether the PIN is correct or not. default: false

Event Messages

None

14.2.4 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 2.0)	Type	Required
{		
"presentAlgorithm": "presentClear",	string	✓
"chipProtocol": "chipT0",	string	✓
"chipData": "Y2hpcCBkYXRhIHRvIHNl ...",	string	✓
"algorithmData": {	object	✓
"pinPointer": 0,	integer	✓
"pinOffset": 0	integer	✓
}		
}		

Properties

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 2.0)	Type	Required
{		
"errorCode": "accessDenied",	string, null	
"chipProtocol": "chipT0",	string, null	

Payload (version 2.0)	Type	Required
"chipData": "Y2hpcCBkYXRhIHJlY2Vp ..."	string, null	
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

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. This value is null if there is no data returned from the chip.

default: null

chipData

The data returned from the chip. This value is null if there is no data returned from the chip.

Property value constraints:

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

default: null

Event Messages

None

14.2.5 PinPad.Reset

Sends a service reset to the Service.

Command Message

Payload (version 2.0)
This message does not define any properties.

Completion Message

Payload (version 2.0)
This message does not define any properties.

Event Messages

None

14.2.6 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 2.0)	Type	Required
{		
"maintainPIN": false	boolean	
}		

Properties

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 2.0)
This message does not define any properties.

Event Messages

None

14.2.7 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 2.0)	Type	Required
{		
"customerData": "9385527846382726",	string, null	
"padding": 2,	integer	
"format": "ibm3624",	string	✓
"key": "PinKey01",	string, null	
"xorData": "0123456789ABCDEF",	string, null	
"secondEncKey": "Key01",	string, null	
"cryptoMethod": "ecb"	string, null	
}		

Properties

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 value is null.

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,}$
```

default: null

padding

Specifies the padding character. This property is ignored for PIN block formats with fixed, sequential or random padding.

Property value constraints:

```
minimum: 0
maximum: 15
```

default: 15

format

Specifies the format of the PIN block. For a list of valid values see pinFormats.

Properties
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. default: null
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 property is null, 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 null) this is ignored. Property value constraints: `pattern: ^[0-9a-fA-F]{2,}?$` default: null
secondEncKey Specifies the key used to format the once encrypted formatted PIN, this property can be null 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. default: null
cryptoMethod This specifies the cryptographic method to be used for this command, this property is null 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*. default: null

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "keyNotFound"`	string, null	
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. The following values are possible: • `keyNotFound` - The specified key was not found. • `accessDenied` - The encryption module is either not initialized or not ready for any vendor specific reason. • `keyNoValue` - The specified key name was found but the corresponding key value has not been loaded. • `useViolation` - The use specified by keyUsage is not supported. • `noPin` - The PIN has not been entered was not long enough or has been cleared. • `formatNotSupported` - The specified format is not supported. • `invalidKeyLength` - The length of keyEncKey or key is not supported by this key or the length of an encryption key is not compatible with the encryption operation required. default: null

Event Messages

None

14.2.8 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 2.0)	Type	Required
{		
"customerData": "9385527846382726",	string, null	
"padding": 2,	integer	
"format": "ibm3624",	string	✓
"key": "PinKey01",	string, null	
"xorData": "0123456789ABCDEF",	string, null	
"secondEncKey": "Key01",	string, null	
"cryptoMethod": "ecb"	string, null	
}		

Properties

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 value is null.

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,}$
```

default: null

padding

Specifies the padding character. This property is ignored for PIN block formats with fixed, sequential or random padding.

Property value constraints:

```
minimum: 0
maximum: 15
```

default: 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.

default: null

Properties
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 property is null, 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 null) this is ignored. Property value constraints: `pattern: ^[0-9a-fA-F]{2,}?$` default: null
secondEncKey Specifies the key used to format the once encrypted formatted PIN, this property can be null 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. default: null
cryptoMethod This specifies the cryptographic method to be used for this command, this property is null 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*. default: null

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "keyNotFound",`	string, null	
`"pinBlock": "UGluYmxvY2sgZGF0YQ=="`	string, null	
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null

Properties

pinBlock
The encrypted PIN block. This value is null if there is no PIN block.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}$`
`format: base64`
default: null

Event Messages

- KeyManagement.DUKPTKSNEvent

15. 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

The XFS4IoT Printer interface is implemented around a forms model which also standardizes the basic document definition.

15.1 General Information

15.1.1 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.

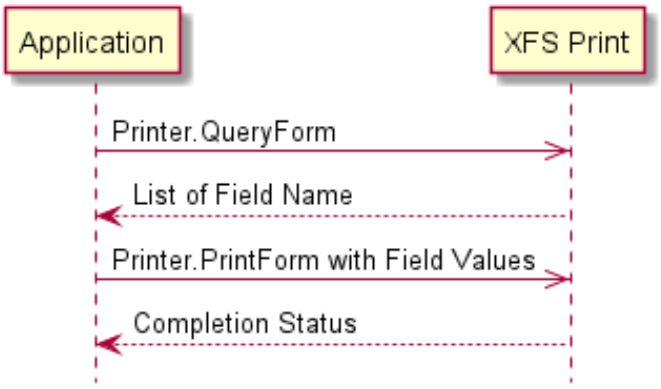
The specification refers to the terms paper and media. When the term paper is used this refers to paper that is situated in a paper supply attached to the device. The term media is used for media that is inserted by the customer (e.g., material that is scanned) or that is issued to the customer (e.g., 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.

15.1.2 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.

15.1.3 Command Overview

The basic operation of the print devices is managed using the two primary commands:

- **Printer.GetQueryForm** This command retrieves the form header information, and the list of fields.
- **Printer.PrintForm** This command includes as parameter data the name of the form to select and the required field data values.

This approach combines in the most efficient manner the four logical steps required to print a form:

- Selecting a document form object.
- Querying the service for the list of fields.
- Supplying the data for each field.
- Issuing the print command.

By using *Printer.GetQueryForm* 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).

15.1.4 Form, Sub-Form, Field, Frame, Table and Media Definitions

This section outlines the format of the definitions of forms, the fields within them, optional tables and fields within the form, and the media on which they are printed.

Definition Syntax

The syntactic rules for form, field and media definitions are as follows:

- **White space**
  Space, Tab
- **Line continuation**
  Backslash (\)
- **Line termination**
  CR, LF, CR/LF; line termination ends a "keyword section" (a keyword and its value[s]).
- **Keywords**
  Must be all upper case.
- **Names**
  Field, media 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.

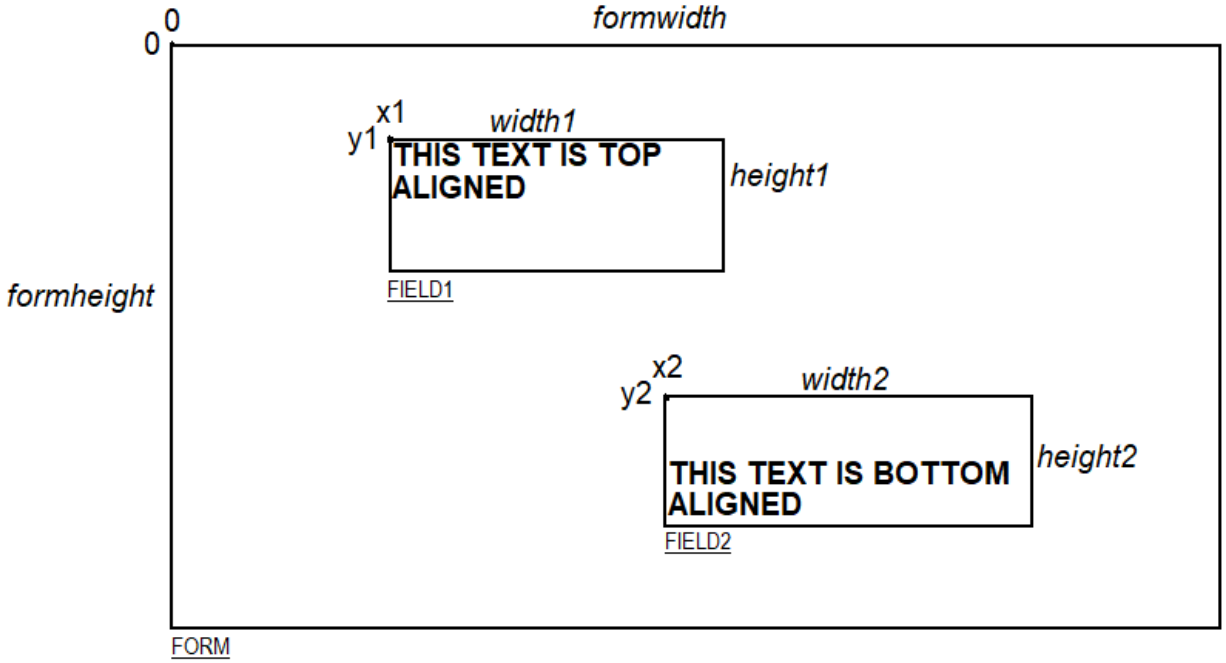
Keyword	Nested Keyword	Required	Names	Notes
	HEADER		*N*, *N-N* or *ALL*	This field is either a form or subform header field. **N** represents a form/subform page number (relative to 0) the header field is to print within. **N-N** represents a form/subform page number range the header field is to print 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

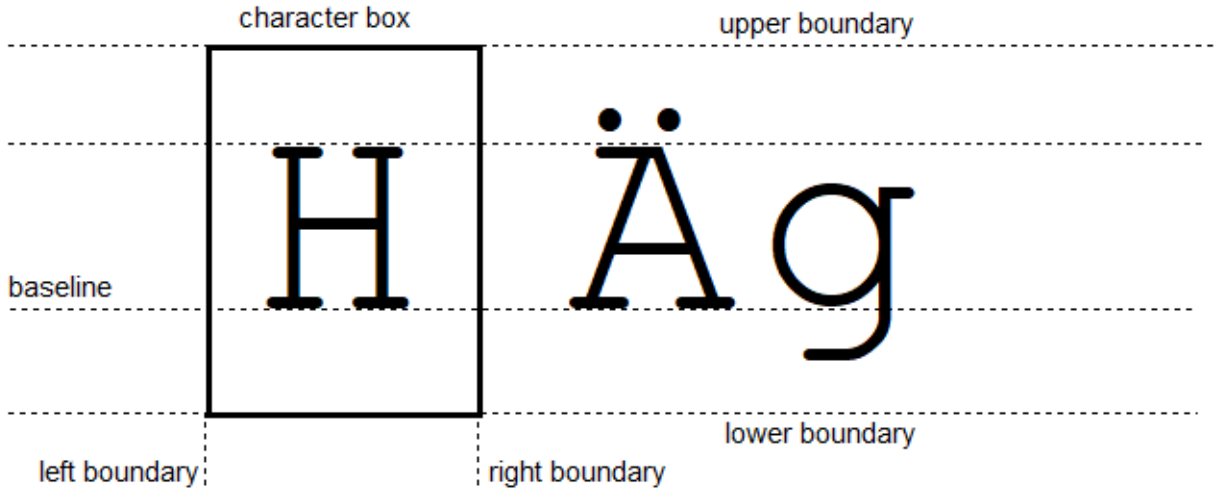
Keyword	Nested Keyword	Required	Names	Notes
	STYLE		*style*	Display attributes as a combination, using the | operator, of the following: NORMAL (default) BOLD ITALIC UNDER (single underline) DOUBLEUNDER (double underline) DOUBLE (double width) TRIPLE (triple width) QUADRUPLE (quadruple width) STRIKETHROUGH 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		✓		

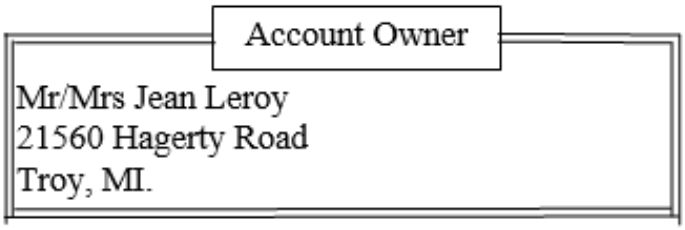
Keyword	Nested Keyword	Required	Names	Notes
	POSITION	✓	*X,Y* or *X,Y,Z*	**X** - Horizontal position of top left corner of the frame (relative to left side of form/subform). **Y** or **Y,Z** - Vertical position of top left corner of frame (relative to top of form/subform). Format **Y,Z** is used to indicate 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

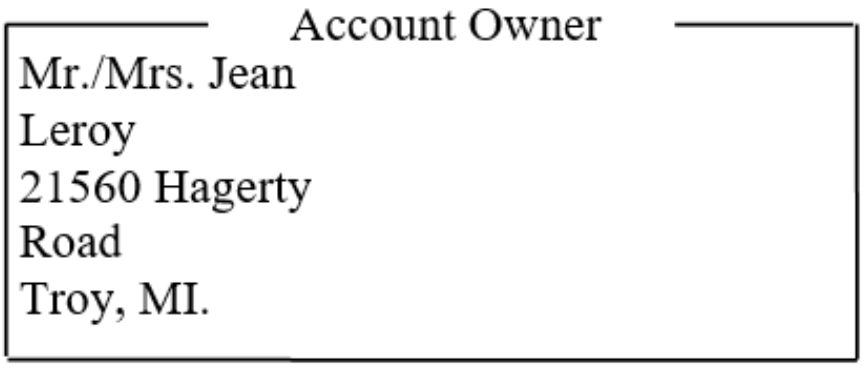
The form:

```
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

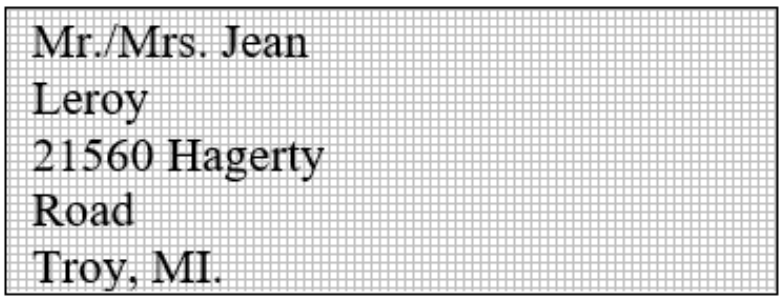
The form:

```
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

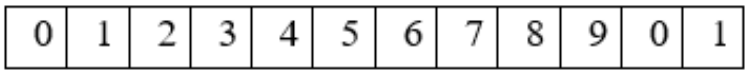
The form:

```
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:



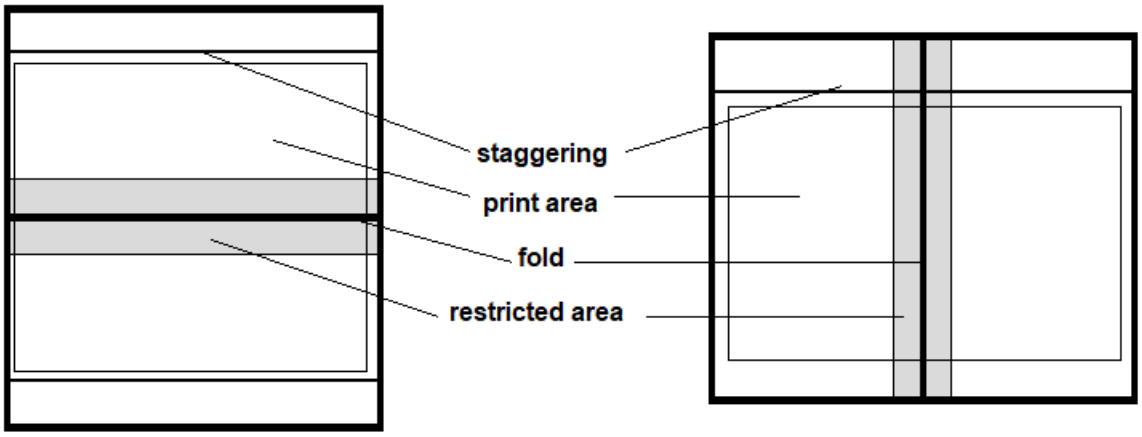
Media Definition

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.

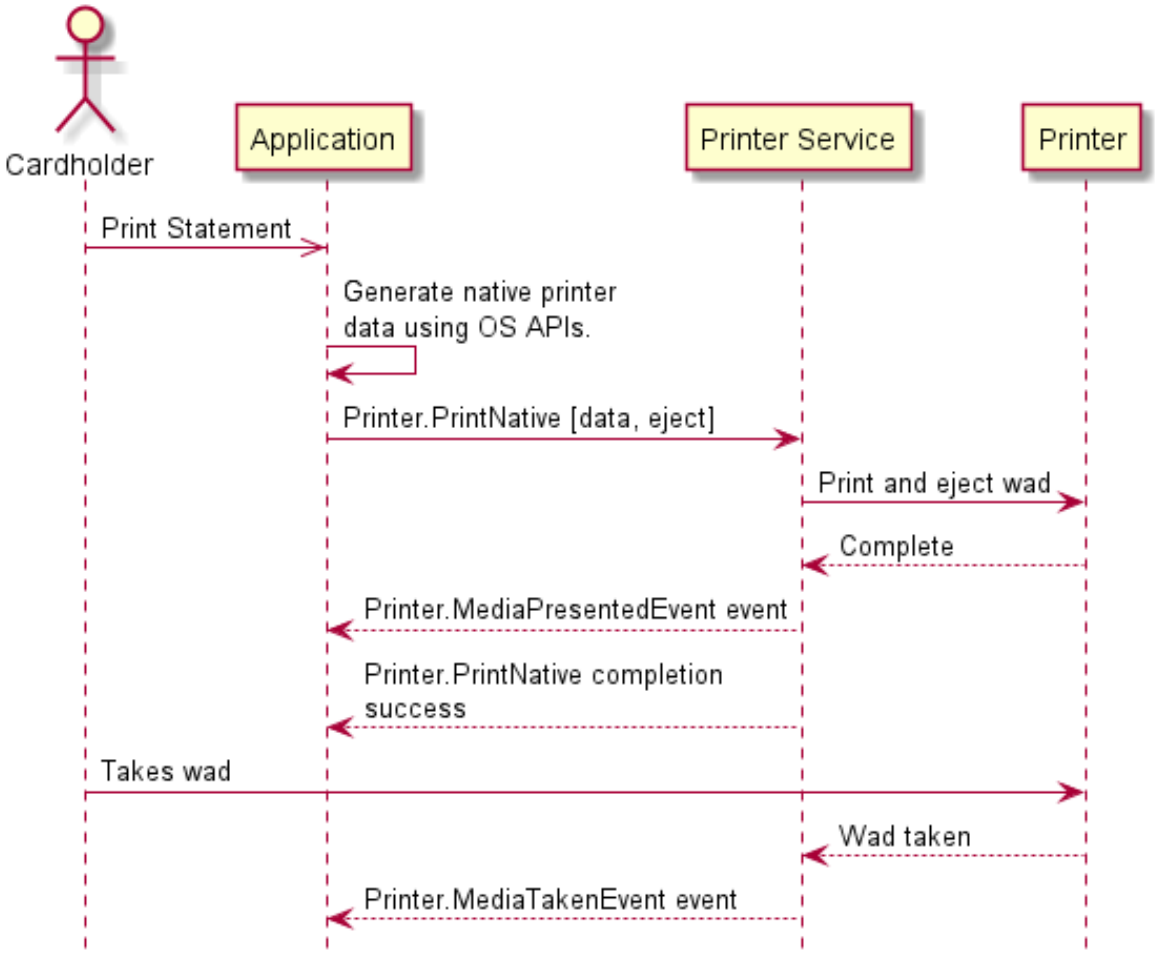
15.1.5 Command and Event Flows during Single and Multi-Page / Wad Printing

It is possible to print a number of pages or bunches of pages (wads) through the Service. The following sections describe how this is achieved.

Single Page / Single Wad Printing With Immediate Media Control

This 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. Printer.PrintNative is used as the example:

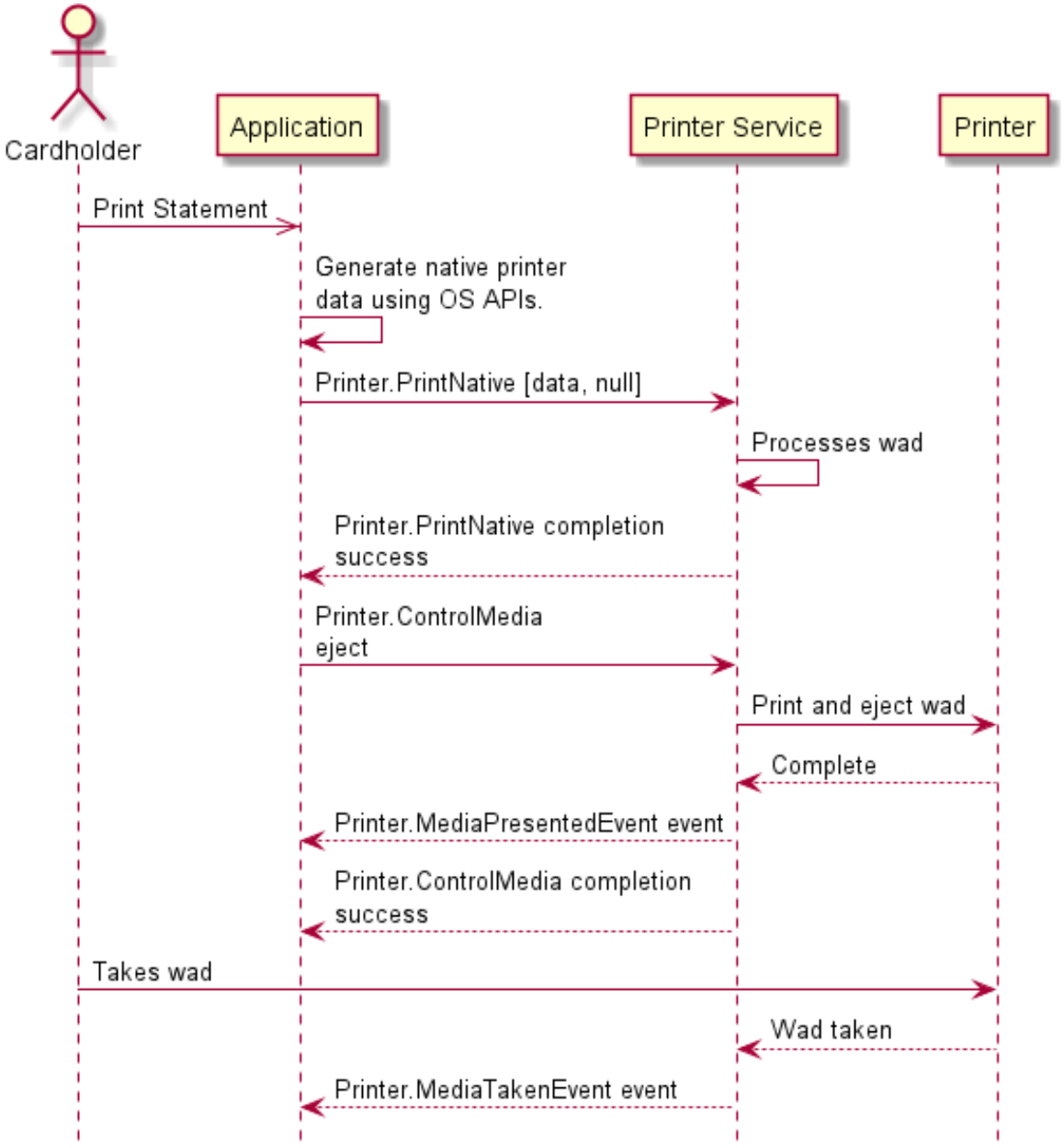
- A single page or single wad of pages is presented.
- The mediaPresented capability is true (indicates that the Printer.MediaPresentedEvent event can be generated).
- The mediaControl in the command data is set to *eject*.



Single Page / Single Wad Printing With Separate Media Control

This 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. The Printer.PrintNative command is used as an example:

- 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 null.
- The media is presented to the user through a Printer.ControlMedia command, with the mediaControl property set to *eject*.

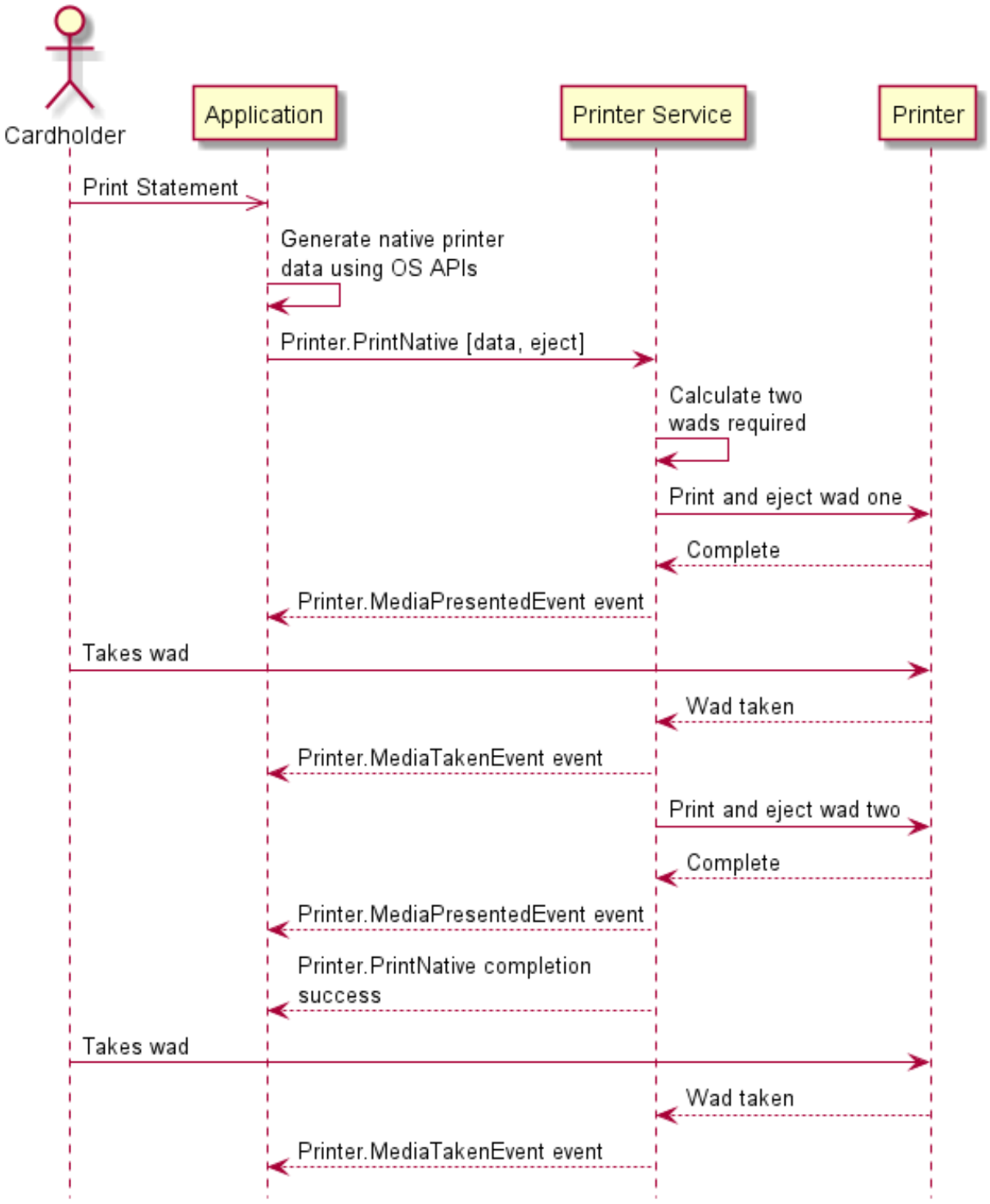


Multi Page / Multi Wad Printing With Immediate Media Control

This 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. Printer.PrintNative is used as the example:

- 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 in the command data is set to *eject*.
- The previous page/wad must be removed before subsequent pages/wads can be presented.

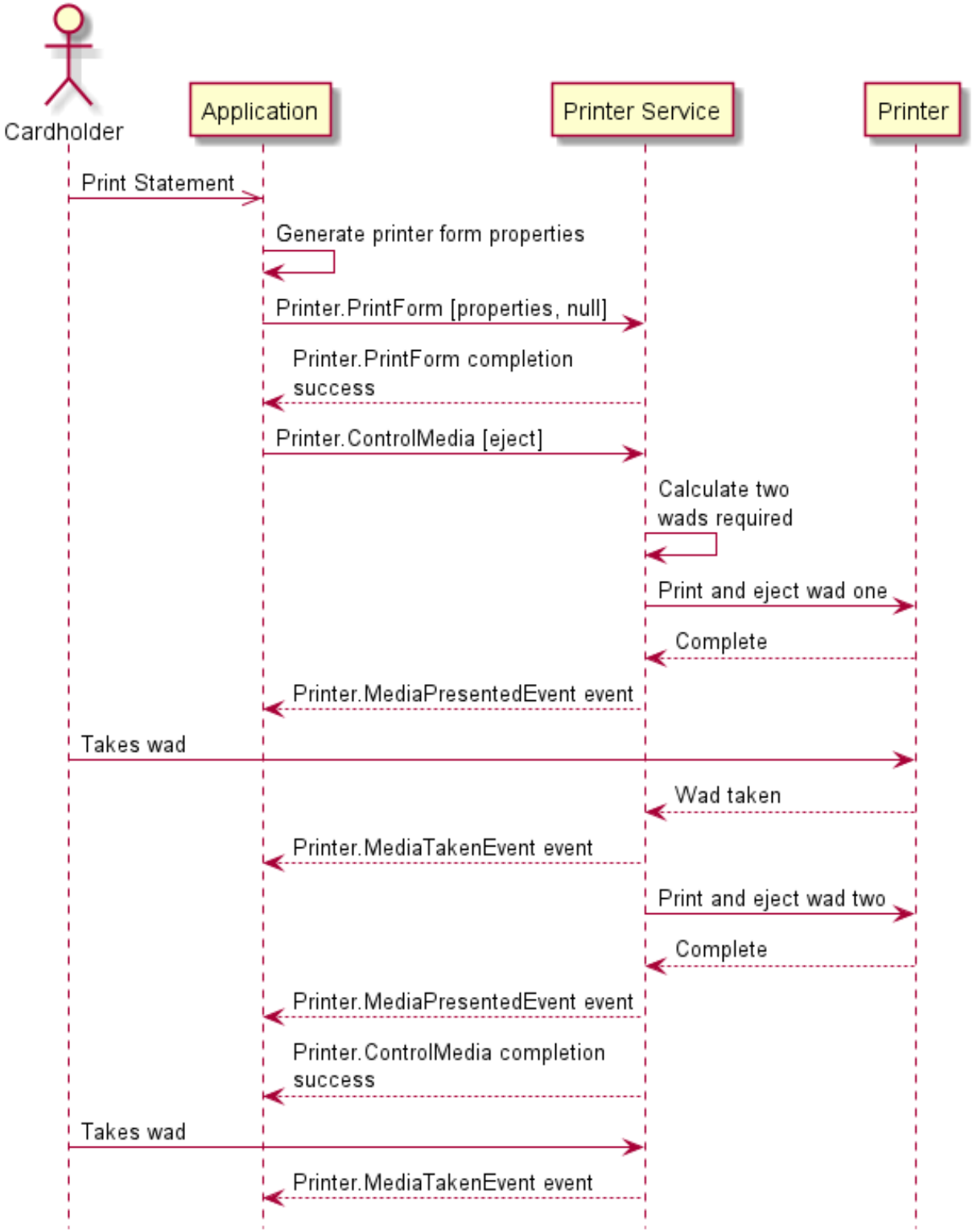


Multi Page / Multi Wad Printing With Separate Media Control

This 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. Printer.PrintForm is used as the example:

- 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 null.

- 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.



15.2 Command Messages

15.2.1 Printer.GetFormList

This command is used to retrieve the list of forms available on the device.

Command Message

Payload (version 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"formList": ["Form1", "Form2"]`	array (string), null	
`}`		

Properties

formList
The list of form names. This will be null if no forms are available.
default: null

Event Messages

None

15.2.2 Printer.GetMediaList

This command is used to retrieve the list of media definitions available on the device.

Command Message

Payload (version 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"mediaList": ["Media1", "Media2"]`	array (string), null	
`}`		

Properties
mediaList The list of media definition names. This will be null if no media definitions are available. default: null

Event Messages

None

15.2.3 Printer.GetQueryForm

This command is used to retrieve details of the definition of a specified form.

Command Message

Payload (version 2.0)	Type	Required
{		
"formName": "example form"	string	✓
}		

Properties

formName
The form name for which to retrieve details.

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "formNotFound",	string, null	
"formName": "Form 1",	string	✓
"base": "na",	string	
"unitX": 0,	integer	
"unitY": 0,	integer	
"width": 0,	integer	
"height": 0,	integer	
"alignment": "na",	string	
"orientation": "na",	string	
"offsetX": 0,	integer	
"offsetY": 0,	integer	
"versionMajor": 0,	integer, null	
"versionMinor": 0,	integer, null	
"userPrompt": "User prompt1",	string, null	
"fields": ["Field1", "Field2"]	array (string), null	
}		

Properties

errorCode
Specifies the error code if applicable, otherwise null. The following values are possible:

- `formNotFound` - The specified form cannot be found.
- `formInvalid` - The specified form is invalid.

default: null

formName
Specifies the name of the form.

Properties
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. • `na` - Not applicable as the specified form cannot be found or is invalid. default: "na"
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` default: 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` default: 0
width Specifies the width of the form in terms of the base horizontal resolution. Property value constraints: `minimum: 0` default: 0
height Specifies the height of the form in terms of the base vertical resolution. Property value constraints: `minimum: 0` default: 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. • `na` - Not applicable as the specified form cannot be found or is invalid. default: "na"
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. • `na` - Not applicable as the specified form cannot be found or is invalid. default: "na"
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. Property value constraints: `minimum: 0` default: 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. Property value constraints: `minimum: 0` default: 0
versionMajor Specifies the major version of the form. This is null if the version is not specified in the form. Property value constraints: `minimum: 0` default: null
versionMinor Specifies the minor version of the form. This is null if the version is not specified in the form. Property value constraints: `minimum: 0` default: null
userPrompt The user prompt string. This will be null if the form does not define a value for the user prompt. default: null
fields The field names. This will be null if the specified form cannot be found or is invalid. default: null

Event Messages

None

15.2.4 Printer.GetQueryMedia

This command is used to retrieve details of the definition of a specified media.

Command Message

Payload (version 2.0)	Type	Required
{		
"mediaName": "example media"	string	✓
}		

Properties

mediaName
The media name for which to retrieve details.

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "mediaNotFound",	string, null	
"mediaType": "generic",	string, null	
"base": "inch",	string, null	
"unitX": 0,	integer	
"unitY": 0,	integer	
"sizeWidth": 0,	integer	
"sizeHeight": 0,	integer	
"pageCount": 0,	integer, null	
"lineCount": 0,	integer, null	
"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, null	
"foldType": "none",	string, null	
"paperSources": {	object, null	
"upper": false,	boolean	
"lower": false,	boolean	
"external": false,	boolean	
"aux": false,	boolean	
"aux2": false,	boolean	
"park": false,	boolean	
"*exampleProperty1*": false,	boolean	

Payload (version 2.0)	Type	Required
"*exampleProperty2*": See *paperSources/exampleProperty1*	boolean	
}		
}		

Properties

errorCode

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

- `mediaNotFound` - The specified media definition cannot be found.
- `mediaInvalid` - The specified media definition is invalid.

default: null

mediaType

Specifies the type of media as one of the following: This property is null if the specified media definition cannot be found or is invalid.

- `generic` - The media is generic, i.e. a single sheet.
- `passbook` - The media is a passbook.
- `multipart` - The media is a multi-part.

default: null

base

Specifies the base unit of measurement of the form and can be one of the following values: This property is null if the specified media definition cannot be found or is invalid.

- `inch` - The base unit is inches.
- `mm` - The base unit is millimeters.
- `rowcolumn` - The base unit is rows and columns.

default: null

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
```

default: 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
```

default: 0

sizeWidth

Specifies the width of the media in terms of the base horizontal resolution.

Property value constraints:

```
minimum: 0
```

default: 0

sizeHeight

Specifies the height of the media in terms of the base vertical resolution.

Property value constraints:

```
minimum: 0
```

default: 0

Properties
pageCount Specifies the number of pages in media of type *passbook*. This will be null if not applicable. Property value constraints: `minimum: 0` default: null
lineCount Specifies the number of lines on a page for a media of type *passbook*. This will be null if not applicable. Property value constraints: `minimum: 0` default: null
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` default: 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` default: 0
printAreaWidth Specifies the printable area width of the media in terms of the base horizontal resolution. Property value constraints: `minimum: 0` default: 0
printAreaHeight Specifies the printable area height of the media in terms of the base vertical resolution. Property value constraints: `minimum: 0` default: 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` default: 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` default: 0
restrictedAreaWidth Specifies the restricted area width of the media in terms of the base horizontal resolution. Property value constraints: `minimum: 0` default: 0

Properties
restrictedAreaHeight Specifies the restricted area height of the media in terms of the base vertical resolution. Property value constraints: `minimum: 0` default: 0
stagger Specifies the staggering from the top in terms of the base vertical resolution for media of type *passbook*. This will be null if not applicable. Property value constraints: `minimum: 0` default: null
foldType Specified the type of fold for media of type *passbook* as one of the following. This will be null if not applicable. • `none` - Passbook has no fold. • `horizontal` - Passbook has a horizontal fold. • `vertical` - Passbook has a vertical fold. default: null
paperSources Specifies the paper sources to use when printing forms using this media. If null, the paper source is determined by the Service. default: null
paperSources/upper The upper paper source. default: false
paperSources/lower The lower paper source. default: false
paperSources/external The external paper source. default: false
paperSources/aux The auxiliary paper source. default: false
paperSources/aux2 The second auxiliary paper source. default: false
paperSources/park The parking station. default: false
paperSources/exampleProperty1 (example name) The vendor specific paper source. Property name constraints: `pattern: ^[a-zA-Z]([a-zA-Z0-9]*)$` default: false

Event Messages

None

15.2.5 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 2.0)	Type	Required
{		
"formName": "Form 10",	string	✓
"fieldName": "Field 3"	string, null	
}		

Properties

formName
The form name.

fieldName
The name of the field about which to retrieve details. If not specified, then details are retrieved for all fields on the form.
default: null

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "formNotFound",	string, null	
"fields": {	object, null	
"*Field 1*": {	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": "na"	string	
}		
}		
}		

Properties

errorCode
Specifies the error code if applicable, otherwise null. 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.

default: null

Properties
fields Details of the field(s) requested. For each object, the key is the field name. This property is null if the form or field cannot be found or is invalid. default: null
fields/Field 1 (example name) Details of a single field.
fields/Field 1/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` default: 0
fields/Field 1/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/Field 1/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/Field 1/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/Field 1/overflow Specifies how an overflow of field data should be handled as one of the following: • `terminate` - Return an error and terminate printing of the form. • `truncate` - Truncate the field data to fit in the field. • `bestFit` - Fit the text in the field. • `overwrite` - Print the field data beyond the extents of the field boundary. • `wordWrap` - If the field can hold more than one line the text is wrapped around. Wrapping is performed, where possible, by splitting the line on a space character or a hyphen character or any other character which is used to join two words together.
fields/Field 1/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 an empty string if the parameter is not specified in the field definition. default: ""
fields/Field 1/format Format string as defined in the form for this field. This value will be an empty string if the parameter is not specified in the field definition. default: ""

Properties
fields/Field 1/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. • `na` - Not applicable. default: "na"

Event Messages

None

15.2.6 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 2.0)	Type	Required
{		
"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

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.

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.

default: false

Properties
mediaControl/perforate Flush data as per eject, then perforate the media. default: false
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. default: false
mediaControl/skip Flush data as per eject, then skip the media to mark. default: false
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. default: false
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. default: false
mediaControl/stack Flush data as per flush, then move the media item on the internal stacker. default: false
mediaControl/partialCut Flush the data as per flush, then partially cut the media. default: false
mediaControl/alarm Cause the printer to ring a bell, beep, or otherwise sound an audible alarm. default: false
mediaControl/forward Flush the data as per flush, then turn one page forward. default: false
mediaControl/backward Flush the data as per flush, then turn one page backward. default: false
mediaControl/turnMedia Flush the data as per flush, then turn inserted media. default: false
mediaControl/stamp Flush the data as per flush, then stamp on inserted media. default: false
mediaControl/park Park the media in the parking station. default: false
mediaControl/expel Flush the data as per flush, then throw the media out of the exit slot. default: false

Properties
mediaControl/ejectToTransport Flush the data as per flush, then move the media to a position on the transport just behind the exit slot. default: false
mediaControl/rotate180 Flush the data as per flush, then rotate media 180 degrees in the printing plane. default: false
mediaControl/clearBuffer Clear any data that has not yet been physically printed from previous Printer.PrintForm or Printer.PrintNative commands. default: false

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "noMediaPresent"`	string, null	
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null

Event Messages

- Printer.MediaPresentedEvent

15.2.7 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 2.0)	Type	Required
{		
"formName": "Form1",	string	✓
"mediaName": "Media1",	string, null	
"alignment": "formDefinition",	string	✓
"offsetX": 0,	integer	
"offsetY": 0,	integer	
"resolution": "low",	string	✓
"mediaControl": {	object, null	
"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	✓
"*Field 1*": "Field Data"	string	
},		

Payload (version 2.0)	Type	Required
`"paperSource": "lower"`	string	
`}`		

Properties

formName
The form name.

mediaName
The media name. If no media definition applies, this should be null.
default: null

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`
default: 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`
default: 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*. This property is null if no actions required.
default: null

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.
default: false

Properties
mediaControl/perforate Flush data as per eject, then perforate the media. default: false
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. default: false
mediaControl/skip Flush data as per eject, then skip the media to mark. default: false
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. default: false
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. default: false
mediaControl/stack Flush data as per flush, then move the media item on the internal stacker. default: false
mediaControl/partialCut Flush the data as per flush, then partially cut the media. default: false
mediaControl/alarm Cause the printer to ring a bell, beep, or otherwise sound an audible alarm. default: false
mediaControl/forward Flush the data as per flush, then turn one page forward. default: false
mediaControl/backward Flush the data as per flush, then turn one page backward. default: false
mediaControl/turnMedia Flush the data as per flush, then turn inserted media. default: false
mediaControl/stamp Flush the data as per flush, then stamp on inserted media. default: false
mediaControl/park Park the media in the parking station. default: false
mediaControl/expel Flush the data as per flush, then throw the media out of the exit slot. default: false

Properties
mediaControl/ejectToTransport Flush the data as per flush, then move the media to a position on the transport just behind the exit slot. default: false
mediaControl/rotate180 Flush the data as per flush, then rotate media 180 degrees in the printing plane. default: false
mediaControl/clearBuffer Clear any data that has not yet been physically printed from previous Printer.PrintForm or Printer.PrintNative commands. default: false
fields An object containing one or more fields.
fields/Field 1 (example name) Property 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.
paperSource Specifes the paper source to be used. For commands which print, 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. • `any` - Use any paper source, it is determined by the service. • `<paper source identifier>` - The vendor specific paper source. Property value constraints: `pattern: ^upper$\|^lower$|^external$\|^aux$|^aux2$\|^park$|^any$\|^[a-zA-Z]([a-zA-Z0-9]*)$` default: "any"

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "formNotFound"`	string, null	
`}`		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

Event Messages

- Printer.NoMediaEvent
- Printer.MediaInsertedEvent
- Printer.FieldErrorEvent
- Printer.FieldWarningEvent
- Printer.MediaPresentedEvent
- Printer.MediaRejectedEvent

15.2.8 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 2.0)	Type	Required
{		
"inputData": "no",	string	✓
"data": "UmF3RGF0YQ=="	string	✓
}		

Properties

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 2.0)	Type	Required
{		
"errorCode": "shutterFail",	string, null	
"data": "UmF3RGF0YQ=="	string	
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

Properties

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

Event Messages

- Printer.MediaPresentedEvent

15.2.9 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 2.0)	Type	Required
{		
"data": "UmF3RGF0YQ==",	string	✓
"mediaControl": {	object, null	
"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

data

The data to be printed.

Property value constraints:

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

Properties
mediaControl Specifies the manner in which the media should be handled after each page is printed. If null or 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. default: null
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. default: false
mediaControl/perforate Flush data as per eject, then perforate the media. default: false
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. default: false
mediaControl/skip Flush data as per eject, then skip the media to mark. default: false
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. default: false
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. default: false
mediaControl/stack Flush data as per flush, then move the media item on the internal stacker. default: false
mediaControl/partialCut Flush the data as per flush, then partially cut the media. default: false
mediaControl/alarm Cause the printer to ring a bell, beep, or otherwise sound an audible alarm. default: false
mediaControl/forward Flush the data as per flush, then turn one page forward. default: false
mediaControl/backward Flush the data as per flush, then turn one page backward. default: false
mediaControl/turnMedia Flush the data as per flush, then turn inserted media. default: false

Properties
mediaControl/stamp Flush the data as per flush, then stamp on inserted media. default: false
mediaControl/park Park the media in the parking station. default: false
mediaControl/expel Flush the data as per flush, then throw the media out of the exit slot. default: false
mediaControl/ejectToTransport Flush the data as per flush, then move the media to a position on the transport just behind the exit slot. default: false
mediaControl/rotate180 Flush the data as per flush, then rotate media 180 degrees in the printing plane. default: false
mediaControl/clearBuffer Clear any data that has not yet been physically printed from previous Printer.PrintForm or Printer.PrintNative commands. default: false
paperSource Specifes the paper source to be used. For commands which print, 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. • `any` - Use any paper source, it is determined by the service. • `<paper source identifier>` - The vendor specific paper source. Property value constraints: `pattern: ^upper$\|^lower$|^external$\|^aux$|^aux2$\|^park$|^any$\|^[a-zA-Z]([a-zA-Z0-9]*)$` default: "any"

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "shutterFail"`	string, null	
`}`		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

Event Messages

- Printer.NoMediaEvent
- Printer.MediaInsertedEvent
- Printer.MediaPresentedEvent
- Printer.MediaRejectedEvent

15.2.10 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 2.0)	Type	Required
{		
"formName": "Form1",	string	✓
"fieldNames": ["FieldName1"],	array (string), null	
"mediaName": "MediaName1",	string, null	
"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 2.0)	Type	Required
"rotate180": false,	boolean	
"clearBuffer": false	boolean	
}		
}		

Properties
formName The name of the form.
fieldNames The field names from which to read input data. If this is null, all input fields on the form will be read. default: null
mediaName The media name. If null, no media definition applies. default: null
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. default: false
mediaControl/perforate Flush data as per eject, then perforate the media. default: false
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. default: false
mediaControl/skip Flush data as per eject, then skip the media to mark. default: false
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. default: false
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. default: false
mediaControl/stack Flush data as per flush, then move the media item on the internal stacker. default: false
mediaControl/partialCut Flush the data as per flush, then partially cut the media. default: false

Properties
mediaControl/alarm Cause the printer to ring a bell, beep, or otherwise sound an audible alarm. default: false
mediaControl/forward Flush the data as per flush, then turn one page forward. default: false
mediaControl/backward Flush the data as per flush, then turn one page backward. default: false
mediaControl/turnMedia Flush the data as per flush, then turn inserted media. default: false
mediaControl/stamp Flush the data as per flush, then stamp on inserted media. default: false
mediaControl/park Park the media in the parking station. default: false
mediaControl/expel Flush the data as per flush, then throw the media out of the exit slot. default: false
mediaControl/ejectToTransport Flush the data as per flush, then move the media to a position on the transport just behind the exit slot. default: false
mediaControl/rotate180 Flush the data as per flush, then rotate media 180 degrees in the printing plane. default: false
mediaControl/clearBuffer Clear any data that has not yet been physically printed from previous Printer.PrintForm or Printer.PrintNative commands. default: false

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "formNotFound",	string, null	
"fields": {	object, null	
"*FieldExample*": "Field Example Data"	string	
}		
}		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null
fields An object containing fields read. If no fields were read, this is null. default: null
fields/FieldExample (example name) A property where 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.

Event Messages

- Printer.NoMediaEvent
- Printer.MediaInsertedEvent
- Printer.FieldErrorEvent
- Printer.FieldWarningEvent
- Printer.MediaRejectedEvent

15.2.11 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 2.0)	Type	Required
{		
"frontImageType": "tif",	string, null	
"backImageType": "tif",	string, null	
"frontImageColorFormat": "binary",	string, null	
"backImageColorFormat": "binary"	string, null	
}		

Properties

frontImageType

Specifies the format of the front image returned by this command as one of the following. This can be null if no front image is requested.

- `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.

default: null

backImageType

Specifies the format of the back image returned by this command as one of the following. This can be null if no back image is requested.

- `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.

default: null

frontImageColorFormat

Specifies the color format of the requested front image as one of the following. This can be null if no front image is requested.

- `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.).

default: null

Properties
backImageColorFormat Specifies the color format of the requested back image as one of the following. This can be null if no back image is requested. • `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.). default: null

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "shutterFail",	string, null	
"images": {	object, null	
"front": {	object, null	
"status": "missing",	string, null	
"data": "SKHFFHGOWORIUNNNLSSL ..."	string	
},		
"back": See images/front properties	object, null	
}		
}		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null
images The status and data for each of the requested images. Only requested images are returned. default: null
images/front The front image status and data. default: null
images/front/status Status of data source. This will be null if not supported, otherwise one of the following values: • `ok` - The data is OK. • `missing` - The data source is missing. default: null

Properties

images/front/data
This contains the Base64 encoded image.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]*={0,2}$`
`format: base64`
default: ""

images/back The back image status and data. default: null

Event Messages

- Printer.NoMediaEvent
- Printer.MediaInsertedEvent
- Printer.MediaRejectedEvent

15.2.12 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 2.0)	Type	Required
{		
"base": "inches",	string	✓
"unitX": 0,	integer	
"unitY": 0	integer	
}		

Properties

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
```

default: 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
```

default: 0

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "extentNotSupported",	string, null	
"sizeX": 0,	integer	
"sizeY": 0	integer	
}		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null
sizeX Specifies the width of the media in terms of the base horizontal resolution. Property value constraints: `minimum: 0` default: 0
sizeY Specifies the height of the media in terms of the base vertical resolution. Property value constraints: `minimum: 0` default: 0

Event Messages

- Printer.NoMediaEvent
- Printer.MediaInsertedEvent
- Printer.MediaRejectedEvent

15.2.13 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 2.0)	Type	Required
{		
"binNumber": 1	integer	
}		

Properties

binNumber

The number of the retract bin for which the retract count should be reset to 0. If 0, all bin counts will be set to 0. See retractBins.

Property value constraints:

```
minimum: 0
```

default: 0

Completion Message

Payload (version 2.0)
This message does not define any properties.

Event Messages

None

15.2.14 Printer.Reset

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

The device will attempt to retract or eject any items found anywhere within the device. This may not always be possible because of hardware problems. The Printer.MediaDetectedEvent event will inform the application where items were actually moved to.

Command Message

Payload (version 2.0)	Type	Required
{		
"mediaControl": "unit1"	string	✓
}		

Properties

mediaControl

Specifies the manner in which the media should be handled, as one of the following:

- `eject` - Eject the media.
- `expel` - Throw the media out of the exit slot.
- `unit<retract bin number>` - Retract the media to retract bin number specified. This number has to be between 1 and the number of bins supported by this device.

Property value constraints:

```
pattern: ^eject$|^expel$|^unit[0-9]+$
```

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "shutterFail"	string, null	
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

Event Messages

- Printer.MediaPresentedEvent

15.2.15 Printer.RetractMedia

The media is removed from its present position (media inserted into device, media entering, unknown position) and stored in one of the retract bins. An event is sent if the storage capacity of the specified retract bin is reached. If the bin is already full and the command cannot be executed, an error is returned and the media remains in its present position.

If a retract request is received for a device with no retract capability, the unsupportedCommand error is returned.

Command Message

Payload (version 2.0)	Type	Required
{		
"mediaControl": "unit1"	string	
}		

Properties

mediaControl

Specifies the manner in which the media should be handled, as one of the following:

- `transport` - Retract the media 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.
- `unit<retract bin number>` - Retract the media to retract bin number specified. This number has to be between 1 and the number of bins supported by this device.

Property value constraints:

```
pattern: ^transport$|^unit[0-9]+$
```

default: "transport"

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "noMediaPresent",	string, null	
"result": "unit1"	string	
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

result

Specifies where the media has actually been deposited, as one of the following:

- `transport` - Media was retracted to the transport.
- `nomedia` - No media was retracted.
- `unit<retract bin number>` - Media was retracted to the retract bin specified.

Property value constraints:

```
pattern: ^transport$|^nomedia$|^unit[0-9]+$
```

default: "nomedia"

Event Messages

None

15.2.16 Printer.DispensePaper

This command is used to move paper (which can also be a new passbook) from a paper source into the print position.

Command Message

Payload (version 2.0)	Type	Required
{		
"paperSource": "lower"	string	
}		

Properties

paperSource

Specifes the paper source to be used. For commands which print, 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.
- `any` - Use any paper source, it is determined by the service.
- `<paper source identifier>` - The vendor specific paper source.

Property value constraints:

```
pattern: ^upper$|^lower$|^external$|^aux$|^aux2$|^park$|^any$|^[a-zA-Z]([a-zA-Z0-9]*)$
```

default: "any"

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "paperJammed"	string, null	
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

Event Messages

- Printer.MediaPresentedEvent

15.2.17 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 2.0)	Type	Required
{		
"definition": "FormDefinition1",	string	✓
"overwrite": false	boolean	
}		

Properties

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 2.0)	Type	Required
{		
"errorCode": "formInvalid"	string, null	
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

Event Messages

None

15.2.18 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 2.0)	Type	Required
{		
"upper": false,	boolean	
"lower": false,	boolean	
"aux": false,	boolean	
"aux2": false,	boolean	
"toner": false,	boolean	
"ink": false,	boolean	
"lamp": false	boolean	
}		

Properties
upper The only paper supply or the upper paper supply was replenished. default: false
lower The lower paper supply was replenished. default: false
aux The auxiliary paper supply was replenished. default: false
aux2 The second auxiliary paper supply was replenished. default: false
toner The toner supply was replenished. default: false
ink The ink supply was replenished. default: false
lamp The imaging lamp was replaced. default: false

Completion Message

Payload (version 2.0)
This message does not define any properties.

Event Messages

None

15.2.19 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 property 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 2.0)	Type	Required
{		
"action": "forward",	string	✓
"count": 3	integer	
}		

Properties

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
```

default: 1

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "noMediaPresent"	string, null	
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

Event Messages

None

15.2.20 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 2.0)	Type	Required
{		
"blackMarkMode": false	boolean	✓
}		

Properties

blackMarkMode
Specifies whether black mark detection and associated functionality is enabled.

Completion Message

Payload (version 2.0)
This message does not define any properties.

Event Messages

None

15.3 Event Messages

15.3.1 Printer.MediaPresentedEvent

This event is used to indicate when media has been presented to the customer for removal.

Event Message

Payload (version 2.0)	Type	Required
{		
"wadIndex": 1,	integer	✓
"totalWads": 0	integer	✓
}		

Properties

wadIndex

Specifies the index (starting from 1) 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
```

15.3.2 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 2.0)	Type	Required
{		
"userPrompt": "Enter paper"	string	
}		

Properties

userPrompt

The user prompt from the form definition. This will be an empty string 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.

default: ""

15.3.3 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 2.0)
`This message does not define any properties.`

15.3.4 Printer.FieldErrorEvent

This event specifies that a fatal error has occurred while processing a field.

Event Message

Payload (version 2.0)	Type	Required
{		
"formName": "Form1",	string	✓
"fieldName": "Field1",	string	✓
"failure": "required"	string, null	
}		

Properties

formName
The form name.

fieldName
The field name.

failure
Specifies the type of failure. This property will be null if the form field type is not supported with the device, otherwsie 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.
- `graphic` - The specified graphic image could not be printed.

default: null

15.3.5 Printer.FieldWarningEvent

This event specifies that a non-fatal error has occurred while processing a field.

Event Message

Payload (version 2.0)	Type	Required
{		
"formName": "Form1",	string	✓
"fieldName": "Field1",	string	✓
"failure": "required"	string, null	
}		

Properties

formName
The form name.

fieldName
The field name.

failure
Specifies the type of failure. This property will be null if the form field type is not supported with the device, otherwsie 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.
- `graphic` - The specified graphic image could not be printed.

default: null

15.3.6 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 2.0)	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.

15.4 Unsolicited Messages

15.4.1 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 2.0)
`This message does not define any properties.`

15.4.2 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 2.0)
This message does not define any properties.

15.4.3 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 2.0)	Type	Required
{		
"wadIndex": 1,	integer	✓
"totalWads": 0	integer	✓
}		

Properties

wadIndex

Specifies the index (starting from 1) 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
```

15.4.4 Printer.MediaDetectedEvent

This event is generated when a media is detected in the device during a reset operation.

Unsolicited Message

Payload (version 2.0)	Type	Required
{		
"position": "unit2"	string	✓
}		

Properties

position

Specifies the media position after the reset operation, as one of the following values:

- `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.
- `unit<retract bin number>` - Media was retracted to the retract bin specified. The bin number is between 1 and the number of bins supported by this device.

Property value constraints:

```
pattern: ^present$|^entering$|^jammed$|^unknown$|^expelled$|^unit[0-9]+$
```

15.4.5 Printer.RetractBinStatusEvent

This event specifies that the status of the retract bin holding the retracted media has changed.

Unsolicited Message

Payload (version 2.0)	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.

15.4.6 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 2.0)	Type	Required
{		
"name": "form name",	string	✓
"type": "media"	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.

15.4.7 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 2.0)	Type	Required
{		
"result": "unit1"	string	✓
}		

Properties

result

Specifies where the media has actually been deposited, as one of the following:

- `transport` - Media was retracted to the transport.
- `jammed` - The media is jammed.
- `unit<retract bin number>` - Media was retracted to the retract bin specified.

Property value constraints:

```
pattern: ^transport$|^jammed$|^unit[0-9]+$
```

15.4.8 Printer.RetractBinThresholdEvent

This event specifies that the status of the retract bin holding the retracted media has changed.

Unsolicited Message

Payload (version 2.0)	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.

15.4.9 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 2.0)	Type	Required
{		
"paperSource": "lower",	string	✓
"threshold": "out"	string	✓
}		

Properties

paperSource

Specifies the paper source as one of the following:

- `upper` - The only paper source or the upper paper source, if there is more than one paper supply.
- `lower` - The lower paper source.
- `external` - The external paper.
- `aux` - The auxiliary paper source.
- `aux2` - The second auxiliary paper source.
- `<paper source identifier>` - The vendor specific paper source.

Property value constraints:

```
pattern: ^upper$|^lower$|^external$|^aux$|^aux2$|^[a-zA-Z]([a-zA-Z0-9]*)$
```

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.

15.4.10 Printer.TonerThresholdEvent

This event is used to specify that the state of the toner (or ink) reached a threshold.

Unsolicited Message

Payload (version 2.0)	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.

15.4.11 Printer.LampThresholdEvent

This event is used to specify that the state of the imaging lamp reached a threshold.

Unsolicited Message

Payload (version 2.0)	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.

15.4.12 Printer.InkThresholdEvent

This event is used to specify that the state of the stamping ink reached a threshold.

Unsolicited Message

Payload (version 2.0)	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.

16. 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.

16.1 General Information

16.1.1 References

ID	Description
textterminal-1	ISO/IEC 646 (ASCII)

16.1.2 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		✓		

16.2 Command Messages

16.2.1 TextTerminal.GetFormList

This command is used to retrieve the list of forms available on the device.

Command Message

Payload (version 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"formList": ["Example form1", "Example form2"]`	array (string), null	
`}`		
Properties		
formList Array of the form names. This property is null if no forms were loaded. default: null		

Event Messages

None

16.2.2 TextTerminal.GetQueryForm

This command is used to retrieve details of the definition of a specified form.

Command Message

Payload (version 2.0)	Type	Required
{		
"formName": "Example form"	string	✓
}		

Properties

formName
Contains the form name on which to retrieve details.

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "formNotFound",	string, null	
"formName": "Example form",	string, null	
"width": 0,	integer, null	
"height": 0,	integer, null	
"versionMajor": 0,	integer, null	
"versionMinor": 0,	integer, null	
"fields": ["Field1", "Field2"]	array (string), null	
}		

Properties

errorCode
Specifies the error code if applicable, otherwise null. The following values are possible:

- `formNotFound` - The specified form cannot be found.
- `formInvalid` - The specified form is invalid.

default: null

formName
Specifies the name of the form. This property is null if the form is not loaded.
default: null

width
Specifies the width of the form in columns.
Property value constraints:

```
minimum: 0
```

default: null

height
Specifies the height of the form in rows.
Property value constraints:

```
minimum: 0
```

default: null

Properties
versionMajor Specifies the major version. Property value constraints: `minimum: 0` default: null
versionMinor Specifies the minor version. Property value constraints: `minimum: 0` default: null
fields A list of the field names. This property is null if the specified form is not loaded or no fields were defined. default: null

Event Messages

None

16.2.3 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 2.0)	Type	Required
{		
"formName": "My form name",	string	✓
"fieldName": "My form field"	string, null	
}		

Properties

formName
Specifies the form name.

fieldName
Specifies the name of the field about which to retrieve details. If this property is null, then retrieve details for all fields on the form.
default: null

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "formNotFound",	string, null	
"fields": {	object, null	
"*Field 1*": {	object, null	
"type": "text",	string	
"class": "optional",	string	
"access": {	object, null	
"read": false,	boolean	
"write": true	boolean	
},		
"overflow": "terminate",	string	
"format": "Format 1"	string, null	
},		
"*Field 2*": See *fields/Field 1* properties	object, null	
}		
}		

Properties

errorCode
Specifies the error code if applicable, otherwise null. 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.

default: null

Properties
fields Details of the field(s) requested. The key is the field name and the value contains the details of the fields. This property is null if the specified form is not loaded. default: null
fields/Field 1 (example name) The details of the field definition. default: null
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 '*'. default: "text"
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. default: "optional"
fields/Field 1/access Specifies whether the field is to be used for input, output or both. default: null
fields/Field 1/access/read The field is used for input from the physical device. default: false
fields/Field 1/access/write The field is used for output to the physical device. default: true
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. default: "terminate"
fields/Field 1/format Format string as defined in the form for this field. This value will be null if the parameter is not specified in the field definition. default: null

Event Messages

None

16.2.4 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 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"keys": ["one", "nine"],`	array (string), null	
`"commandKeys": {`	object, null	
`"enter": {`	object	
`"terminate": false`	boolean	
`},`		
`"oem1": See commandKeys/enter properties`	object	
`}`		
`}`		

Properties

keys

String array which contains the printable characters numeric and alphanumeric keys on the Text Terminal Unit, e.g. ["zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine", "A", "B", "C", "a", "b", "c"] if those text terminal input keys are present. This property will be null if no keys supported.

The following prefixed key 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
- `\D` - Any character other than a decimal digit

Property value constraints:

```
pattern: ^(zero|one|two|three|four|five|six|seven|eight|nine|\D)$
minItems: 1
uniqueItems: true
```

default: null

commandKeys

Supporting command keys on the Text Terminal Unit. This property can be null if no command keys supported.

Property value constraints:

```
minProperties: 1
```

default: null

Properties
commandKeys/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]*)$`
commandKeys/enter/terminate The key is a terminate key. default: false

Event Messages

None

16.2.5 TextTerminal.Beep

This command is used to beep at the text terminal unit.

Command Message

Payload (version 2.0)	Type	Required
`{`		
`"beep": {`	object	✓
`"continuous": false,`	boolean	
`"beepType": "keyPress"`	string	
`}`		
`}`		

Properties

beep
Specifies whether the beeper should be turned on or off.

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.

default: "keyPress"

Completion Message

Payload (version 2.0)
`This message does not define any properties.`

Event Messages

None

16.2.6 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 2.0)	Type	Required
{		
"screen": {	object, null	
"positionX": 0,	integer	✓
"positionY": 0,	integer	✓
"width": 1,	integer	✓
"height": 1	integer	✓
}		
}		

Properties

screen
Specify the area of the text terminal unit screen to clear. If this property is null, the whole screen will be cleared.
default: null

screen/positionX
Specifies the horizontal position of the area to be cleared.
Property value constraints:
```
minimum: 0
```

screen/positionY
Specifies the vertical position of the area to be cleared.
Property value constraints:
```
minimum: 0
```

screen/width
Specifies the width position of the area to be cleared.
Property value constraints:
```
minimum: 1
```

screen/height
Specifies the height position of the area to be cleared.
Property value constraints:
```
minimum: 1
```

Completion Message

Payload (version 2.0)
This message does not define any properties.

Event Messages

None

16.2.7 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 2.0)	Type	Required
{		
"resolution": {	object	✓
"sizeX": 0,	integer	✓
"sizeY": 0	integer	✓
}		
}		

Properties

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 2.0)	Type	Required
{		
"errorCode": "resolutionNotSupported"	string, null	
}		

Properties

errorCode
Specifies the error code if applicable, otherwise null. The following values are possible:

- `resolutionNotSupported` - The specified resolution is not supported by the display.

default: null

Event Messages

None

16.2.8 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 2.0)	Type	Required
{		
"formName": "My form",	string	✓
"clearScreen": true,	boolean	
"fields": {	object	✓
"*Field1*": "field",	string	
"*Field2*": See *fields/Field1*	string	
}		
}		

Properties

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.

fields/Field1 (example name)
Field data.

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "formNotFound"	string, null	
}		

Properties

errorCode
Specifies the error code if applicable, otherwise null. 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.

default: null

Event Messages

- TextTerminal.FieldErrorEvent

- TextTerminal.FieldWarningEvent

16.2.9 TextTerminal.ReadForm

This command is used to read data from input fields on the specified form.

The 'enter' key only acts as terminate key when it is pressed in the last read field. When the 'enter' key is pressed in an intermediate field, the cursor moves to the next field and the data entry finishes for the current field. Any other key that terminates input (except cancel), will cause all the fields to be returned in their present state. If cancel terminates input then the command will return the *keyCanceled* error.

The following keys will not be returned in the output property *fields*, but they may affect the field content (note in the following the term field content is used to refer to the data buffer and the display field):

Command key	Meaning
clear	Will clear the field content.
backspace	Will cause the character before the Current Edit Position to be removed from the field content. If 'backspace' is the first key pressed after a field is activated (for any reason other than when the 'backspace' key causes the field to be activated), then the last character in the field content is deleted. If 'backspace' is pressed when the Current Edit Position is at the start of a field, then the previous field is activated. If 'backspace' is the first key pressed after the field is activated as a result of an earlier 'backspace' then no characters are deleted from the field content and the previous field will be activated. It is not possible to navigate backwards past the first field; in this case 'backspace' will have no effect.
doubleZero	Will add a double zero '00' string to the field content. If there is not enough space for all the digits to be added to the field content when the fields's OVERFLOW definition is TERMINATE or TRUNCATE then the excess '0's will be ignored. If the field's OVERFLOW definition is OVERWRITE then all the '0's are added to the field content.
tripleZero	Will add a triple zero '000' string to the field content. If there is not enough space for all the digits to be added to the field content when the field's OVERFLOW definition is TERMINATE or TRUNCATE then the excess '0's will be ignored. If the field's OVERFLOW definition is OVERWRITE then all the '0's are added to the field content.

Command Message

Payload (version 2.0)	Type	Required
{		
"formName": "My form",	string	✓
"fields": ["Field1", "Field2"]	array (string), null	
}		

Properties

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 this property is null, data is read from all input fields on the form in the order they appear in the form (independent of the field screen position).
default: null

Completion Message

Payload (version 2.0)	Type	Required
{		

Payload (version 2.0)	Type	Required
"errorCode": "formNotFound",	string, null	
"fields": {	object, null	
"*Field1*": "123",	string	
"*Field2*": See *fields/Field1*	string	
}		
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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 cancel key.
- `fieldError` - An error occurred while processing a field.

default: null

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. This property is null if no fields were read.

default: null

fields/Field1 (example name)

Field data.

Event Messages

- TextTerminal.FieldErrorEvent
- TextTerminal.FieldWarningEvent

16.2.10 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 2.0)	Type	Required
{		
"mode": "absolute",	string	
"posX": 0,	integer	
"posY": 0,	integer	
"textAttr": {	object, null	
"underline": false,	boolean	
"inverted": false,	boolean	
"flash": false	boolean	
},		
"text": "Text to display"	string	
}		

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*.

default: "absolute"

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
```

default: 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
```

default: 0

textAttr

Specifies the text attributes used for displaying the text. This property is null if not applicable.

default: null

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. default: ""

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "characterSetsData"`	string, null	
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. The following values are possible: • `characterSetsData` - The character set(s) supported by the Service is inconsistent with *text*. default: null

Event Messages

None

16.2.11 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 2.0)	Type	Required
{		
"numOfChars": 0,	integer	
"mode": "absolute",	string	
"posX": 0,	integer	
"posY": 0,	integer	
"echoMode": "text",	string	
"echoAttr": {	object, null	
"underline": false,	boolean	
"inverted": false,	boolean	
"flash": false	boolean	
},		
"visible": true,	boolean	
"flush": false,	boolean	
"autoEnd": true,	boolean	
"activeKeys": ["one", "nine"],	array (string), null	
"activeCommandKeys": {	object, null	
"*enter*": {	object	
"terminate": false	boolean	
},		
"*oem1*": See *activeCommandKeys/enter* properties	object	
}		
}		

Properties

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
```

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*.

default: "absolute"

Properties
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` default: 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` default: 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 this property is null then the text will be displayed as normal text. Property value constraints: `minProperties: 1` default: null
echoAttr/underline The displayed text will be underlined. default: false
echoAttr/inverted The displayed text will be inverted. default: false
echoAttr/flash The displayed text will be flashing. default: false
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). default: 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

Properties

activeKeys

Specifying the numeric and alphanumeric keys on the Text Terminal Unit, e.g. ["one", "two", "A", "B", "a", "b"] 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. ["one", "two", "A", "a", "B", "b"]).

For devices not having a shift key, specifying either the upper case only (e.g. ["one", "two", "A", "B"]), or specifying both the upper and lower case of a particular letter (e.g. ["one", "two", "A", "a", "B", "b"]), 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. ["one", "two", "a", "b"]) produces a key invalid error. This property is null if no keys of this type are active keys.

See predefined keys.

Property value constraints:

```
pattern: ^(zero|one|two|three|four|five|six|seven|eight|nine|\D)$
minItems: 1
uniqueItems: true
```

default: null

activeCommandKeys

Specifying the command keys which are active during the execution of the command. This property is null if no keys of this type are active keys.

Property value constraints:

```
minProperties: 1
```

default: null

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|arrowLe
ft|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 2.0)	Type	Required
{		
"errorCode": "keyInvalid",	string, null	
"input": "12345"	string, null	
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

input

Specifies a string containing all the printable characters (numeric and alphanumeric) read from the text terminal unit key pad. This property is null if no characters are read.

default: null

Event Messages

- TextTerminal.KeyEvent

16.2.12 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 2.0)
This message does not define any properties.

Completion Message

Payload (version 2.0)
This message does not define any properties.

Event Messages

None

16.2.13 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 2.0)	Type	Required
{		
"activeKeys": ["one", "nine"],	array (string), null	
"activeCommandKeys": {	object, null	
"enter": {	object	
"terminate": false	boolean	
},		
"oem1": See activeCommandKeys/enter properties	object	
}		
}		

Properties

activeKeys

String array which specifies the alphanumeric keys on the Text Terminal Unit, e.g. ["one", "two", "B", "a", "b"], 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. ["one", "two", "A", "a", "B", "b"]).

For devices not having a shift key, specifying either the upper case only (e.g. ["one", "two", "A", "B"]), or specifying both the upper and lower case of a particular letter (e.g. ["one", "two", "A", "a", "B", "b"]), 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"["one", "two", "a", "b"]) produces a key invalid error.

See predefined keys.

Property value constraints:

```
pattern: ^(zero|one|two|three|four|five|six|seven|eight|nine|\D)$
minItems: 1
uniqueItems: true
```

default: null

activeCommandKeys

Array specifying the command keys which are active during the execution of the next TextTerminal.ReadForm command. This property is null if no active command keys are required.

Property value constraints:

```
minProperties: 1
```

default: null

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 2.0)	Type	Required
`{`		
`"errorCode": "keyInvalid"`	string, null	
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null

Event Messages

None

16.2.14 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 2.0)	Type	Required
{		
"definition": "See form description",	string	✓
"overwrite": false	boolean	
}		

Properties

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 2.0)	Type	Required
{		
"errorCode": "formInvalid"	string, null	
}		

Properties

errorCode

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

- `formInvalid` - The form is invalid.
- `definitionExists` - The specified form definition already exists and *overwrite* was false.

default: null

Event Messages

- TextTerminal.FormLoadedEvent

16.3 Event Messages

16.3.1 TextTerminal.FieldErrorEvent

This event specifies that a fatal error has occurred while processing a field.

Event Message

Payload (version 2.0)	Type	Required
{		
"formName": "Example form",	string	✓
"fieldName": "Field1",	string	✓
"failure": "required"	string	✓
}		

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.

16.3.2 TextTerminal.FieldWarningEvent

This event is used to specify that a non-fatal error has occurred while processing a field.

Event Message

Payload (version 2.0)	Type	Required
{		
"formName": "Example form",	string	✓
"fieldName": "Field1",	string	✓
"failure": "required"	string	✓
}		

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.

16.3.3 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 2.0)	Type	Required
{		
"key": "string",	string, null	
"commandKey": "string"	string, null	
}		

Properties

key
Specifies the command key supported.
See predefined keys.
Property value constraints:

```
pattern: ^(zero|one|two|three|four|five|six|seven|eight|nine|\D)$
```

default: null

commandKey
Specifies the command key supported.
See predefined keys.
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]*)$
```

default: null

16.3.4 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 2.0)	Type	Required
{		
"name": "Form 1"	string	✓
}		
Properties		
name Specifies the name of the form just loaded.		

17. 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.

17.1 Command Messages

17.1.1 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 2.0)	Type	Required
{		
"symbologies": {	object, null	
"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	

Payload (version 2.0)	Type	Required
"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

symbologies

Specifies the sub-set of barcode 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 and can be null. If all symbologies should be accepted then this property should be null.

default: null

symbologies/ean128

GS1-128

default: false

Properties
symbologies/ean8 EAN-8 default: false
symbologies/ean8_2 EAN-8 with 2 digit add-on default: false
symbologies/ean8_5 EAN-8 with 5 digit add-on default: false
symbologies/ean13 EAN-13 default: false
symbologies/ean13_2 EAN-13 with 2 digit add-on default: false
symbologies/ean13_5 EAN-13 with 5 digit add-on default: false
symbologies/jan13 JAN-13 default: false
symbologies/upcA UPC-A default: false
symbologies/upcE0 UPC-E default: false
symbologies/upcE0_2 UPC-E with 2 digit add-on default: false
symbologies/upcE0_5 UPC-E with 5 digit add-on default: false
symbologies/upcE1 UPC-E with leading 1 default: false
symbologies/upcE1_2 UPC-E with leading 1and 2 digit add-on default: false
symbologies/upcE1_5 UPC-E with leading 1and 5 digit add-on default: false
symbologies/upcA_2 UPC-A with2 digit add-on default: false

Properties
symbologies/upcA_5 UPC-A with 5 digit add-on default: false
symbologies/codabar CODABAR (NW-7) default: false
symbologies/itf Interleaved 2 of 5 (ITF) default: false
symbologies/code11 CODE 11 (USD-8) default: false
symbologies/code39 CODE 39 default: false
symbologies/code49 CODE 49 default: false
symbologies/code93 CODE 93 default: false
symbologies/code128 CODE 128 default: false
symbologies/msi MSI default: false
symbologies/plessey PLESSEY default: false
symbologies/std2Of5 STANDARD 2 of 5 (INDUSTRIAL 2 of 5 also) default: false
symbologies/std2Of5Iata STANDARD 2 of 5 (IATA Version) default: false
symbologies/pdf417 PDF-417 default: false
symbologies/microPdf417 MICROPDF-417 default: false
symbologies/dataMatrix GS1 DataMatrix default: false

Properties
symbologies/maxiCode MAXICODE default: false
symbologies/codeOne CODE ONE default: false
symbologies/channelCode CHANNEL CODE default: false
symbologies/telepenOriginal Original TELEPEN default: false
symbologies/telepenAim AIM version of TELEPEN default: false
symbologies/rss GS1 DataBar™ default: false
symbologies/rssExpanded Expanded GS1 DataBar™ default: false
symbologies/rssRestricted Restricted GS1 DataBar™ default: false
symbologies/compositeCodeA Composite Code A Component default: false
symbologies/compositeCodeB Composite Code B Component default: false
symbologies/compositeCodeC Composite Code C Component default: false
symbologies/posiCodeA Posicode Variation A default: false
symbologies/posiCodeB Posicode Variation B default: false
symbologies/triopticCode39 Trioptic Code 39 default: false
symbologies/codablockF Codablock F default: false

Properties
symbologies/code16K Code 16K default: false
symbologies/qrCode QR Code default: false
symbologies/aztec Aztec Codes default: false
symbologies/ukPost UK Post default: false
symbologies/planet US Postal Planet default: false
symbologies/postnet US Postal Postnet default: false
symbologies/canadianPost Canadian Post default: false
symbologies/netherlandsPost Netherlands Post default: false
symbologies/australianPost Australian Post default: false
symbologies/japanesePost Japanese Post default: false
symbologies/chinesePost Chinese Post default: false
symbologies/koreanPost Korean Post default: false

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "barcodeInvalid",	string, null	
"readOutput": [{	array (object), null	
"symbology": "symbologyUnknown",	string	
"barcodeData": "YmFyY29kZSBkYXRh",	string, null	
"symbologyName": "code39"	string, null	

Payload (version 2.0)	Type	Required
`  }]`		
`}`		

Properties

errorCode

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

- `barcodeInvalid` - The read operation could not be completed successfully. The barcode presented was defective or was wrongly read.

default: null

readOutput

An array of barcode data structures, one for each barcode scanned during the read operation. If no barcode was scanned, this property is null.

default: null

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.
- `triopticCode39` - 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.

default: "symbologyUnknown"

readOutput/barcodeData

Contains the Base64 encoded barcode data read from the barcode reader. The format of the data will depend on the barcode symbology read. In most cases this will be an array of bytes containing ASCII numeric digits. However, the format of the data in this 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. If the read operation could not be completed successfully, this will be null.

Property value constraints:

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

default: null

readOutput/symbologyName

A vendor dependent symbology identifier for the symbology recognized. May be null if not applicable.

default: null

Event Messages

None

17.1.2 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 2.0)
This message does not define any properties.

Completion Message

Payload (version 2.0)
This message does not define any properties.

Event Messages

None

18. 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)**.

18.1 General Information

18.1.1 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.

18.1.2 Enrollment

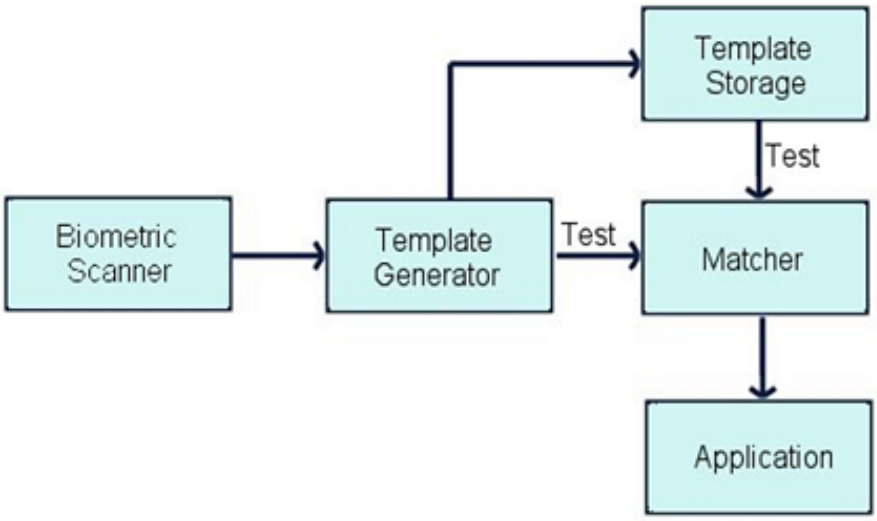
The first time an individual uses a biometric device it is called Enrollment. During enrollment, biometric data from an individual is captured and stored somewhere, for example on a smart card or in a server/host database. Normally the raw biometric data captured will be processed and converted to a 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.

18.1.3 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.

18.1.4 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.

18.1.5 Biometric Data Security

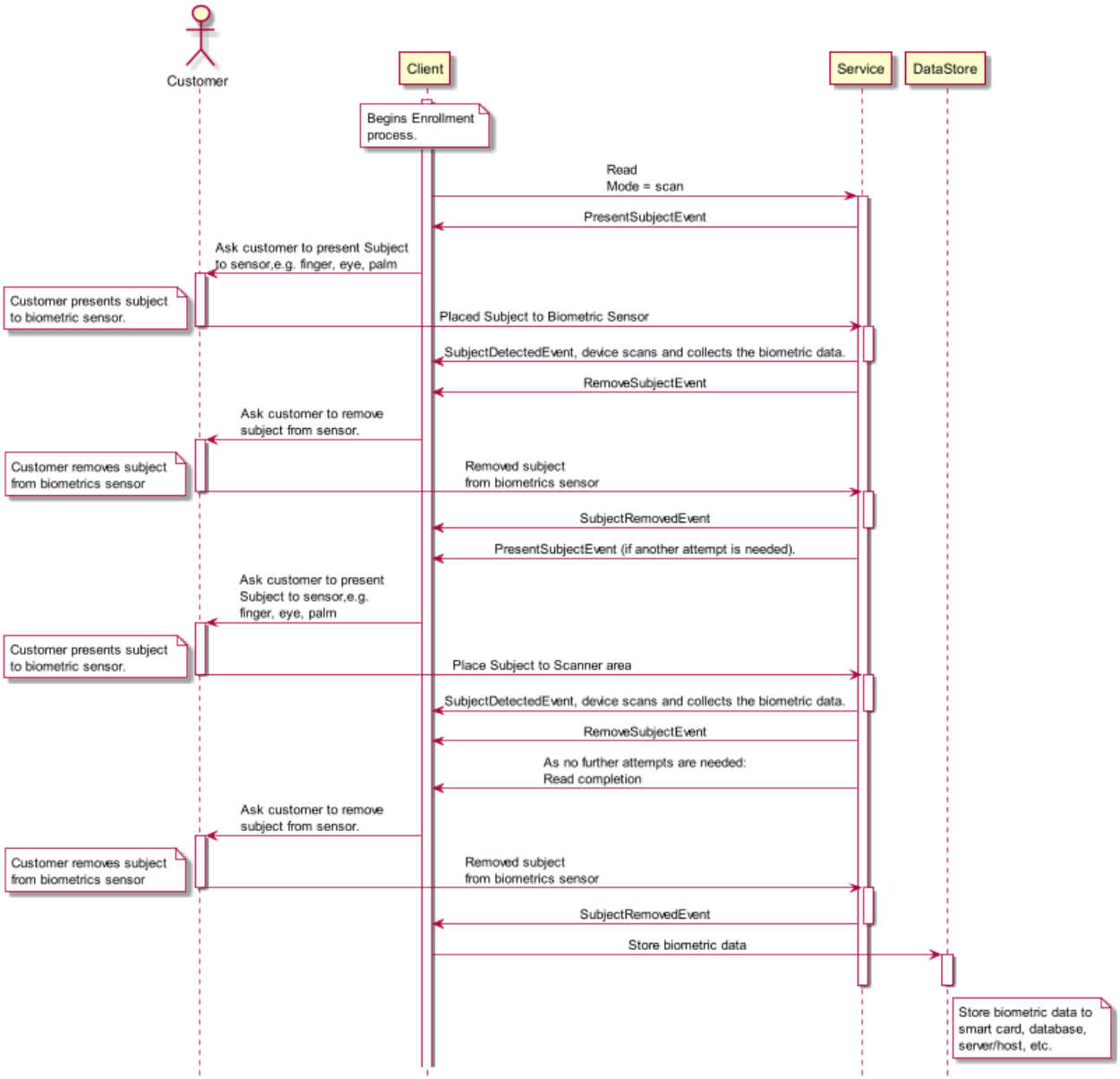
It is recommended that biometric data should be treated with the same strict caution as any other identifying and sensitive information. A well-designed biometric data handling architecture should always be designed to protect against internal tampering, external attacks and other malicious threats. 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.

18.1.6 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.

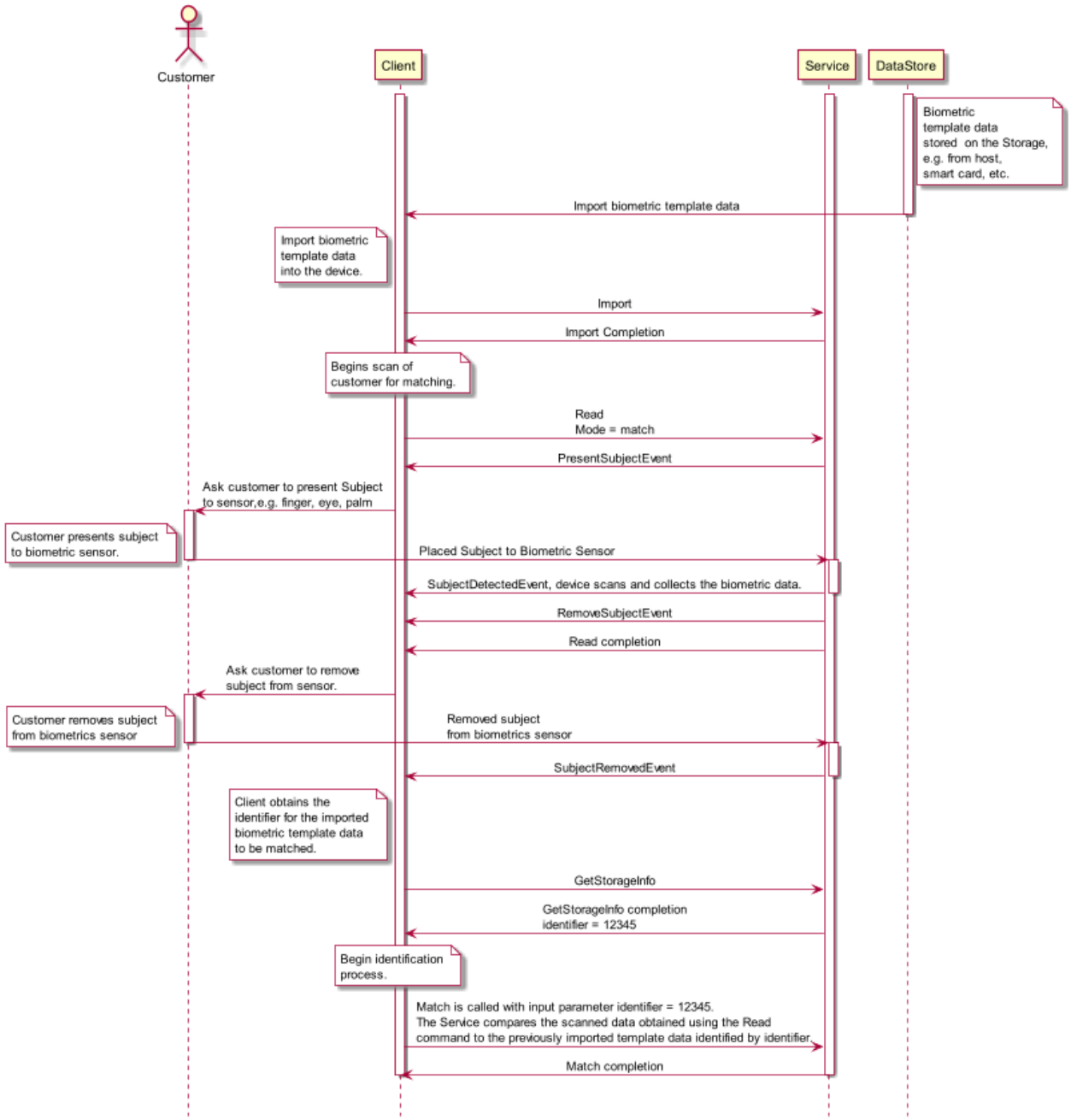


Biometric Match Command Flow – Separate Scan and Match

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.Read is 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.



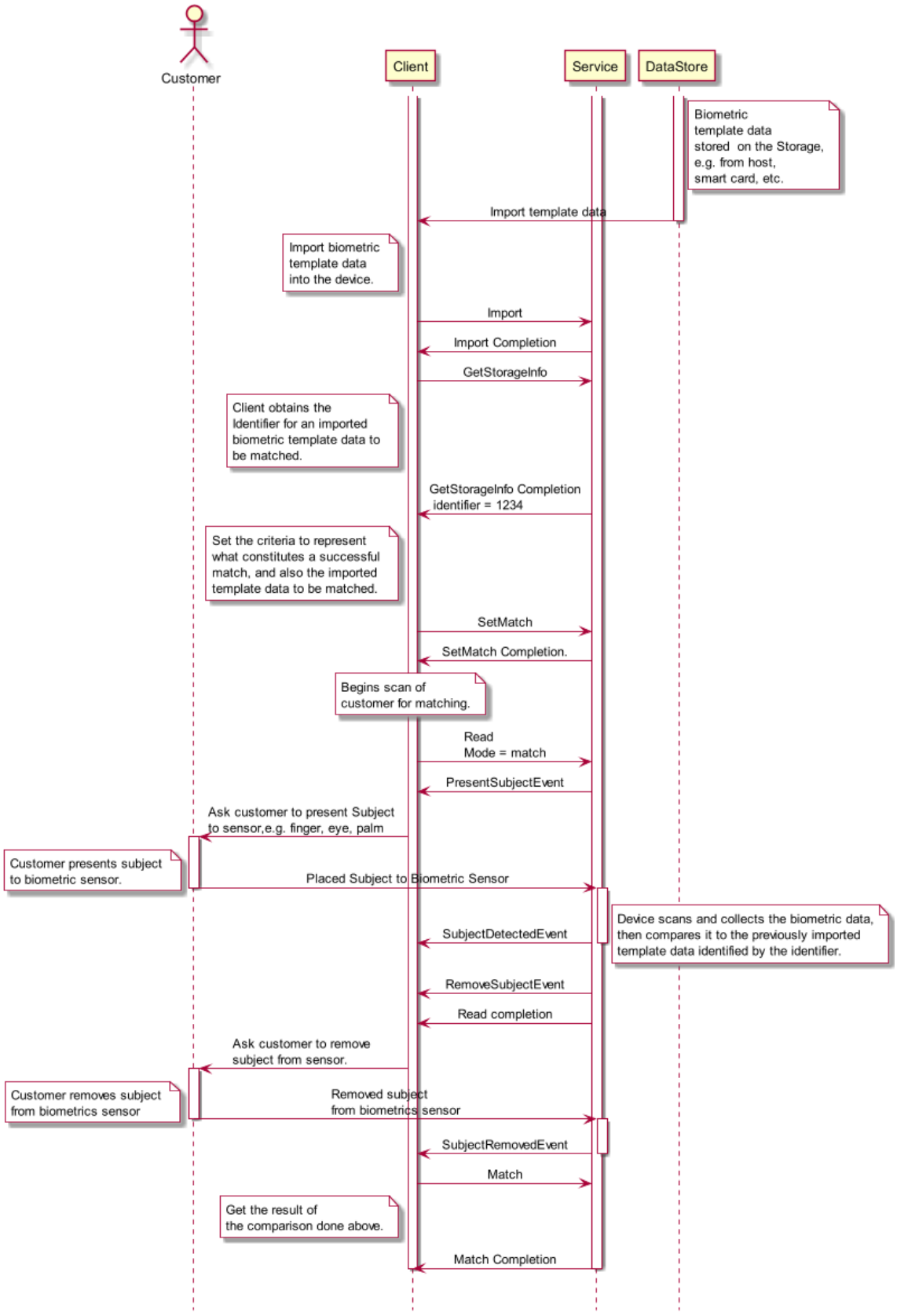
Biometric Match Command Flow – Combined Scan and Match

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.

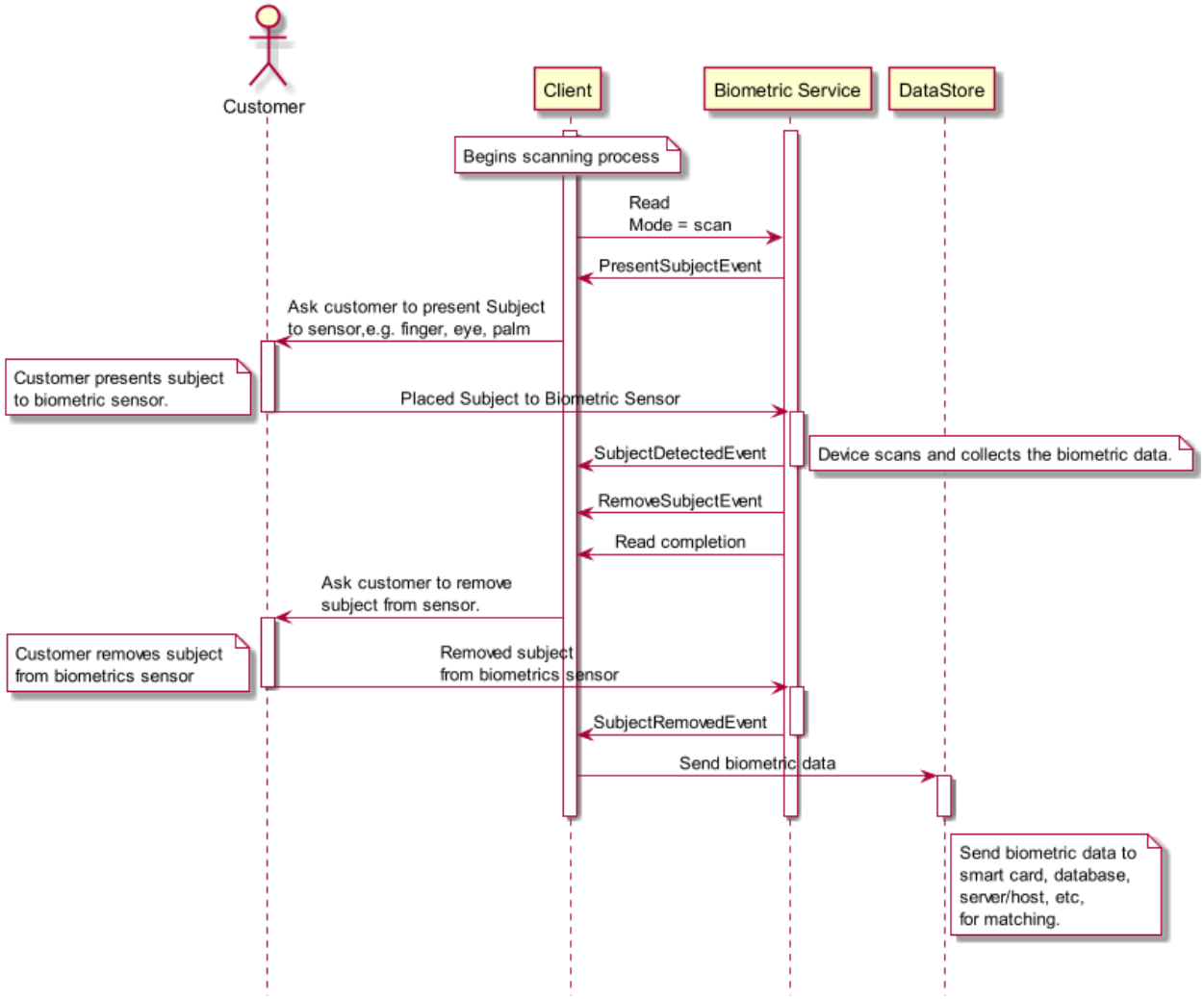
The Biometric.Read, Biometric.SetMatch and Biometric.Match are then used to scan data and compare it with the template identified by *identifier*. In this use case the device performs a combined scan and match operation, therefore the Biometric.SetMatch must be used to set the criteria to be used for 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.

18.2 Command Messages

18.2.1 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 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "noImportedData",	string, null	
"templates": {	object, null	
"*id1*": {	object	
"format": "isoFid",	string	✓
"algorithm": "ecb",	string, null	
"keyName": "Key01"	string, null	
},		
"*id2*": See *templates/id1* properties	object	
}		
}		

Properties

errorCode

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

- `noImportedData` - No data to return. Typically means that no data has been imported using the Biometric.Import.

default: null

templates

A list of biometric templates that were successfully imported. The object name of each biometric data type can be used in the *identifier* property for the Biometric.Match command. If no template data was imported, this property is null.

default: null

templates/id1 (example name)

A unique identifier of the biometric template data.

Property name constraints:

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

Properties
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 null if the biometric data is not encrypted. Available values are described in the encryptionAlgorithms. 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. default: null
templates/id1/keyName Specifies the name of the key that is used to encrypt the biometric data. This property is null if the biometric data is not encrypted. The detailed key information is available through the KeyManagement.GetKeyDetail. default: null

Event Messages

None

18.2.2 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.SubjectDetectedEvent sent for each scanning attempt. The *numCaptures* input parameter specifies how many captures should be attempted, unless it is zero in which case the device itself will determine this. Once this command has successfully captured biometric raw data it will complete with Success.

The Biometric.Read command has two purposes:

Scanning: The biometric data that is captured into the device can be processed into biometric template data and returned as an output parameter for enrollment or storage elsewhere, e.g. on a server or smart card.

Matching: The biometric data that is captured into the device can be used for subsequent matching. Once data has been scanned into the device it can be compared to existing biometric templates that have been imported using the Biometric.Import in order to allow verification or identification of an individual. The matchSupported capability 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 2.0)	Type	Required
`{`		
`"dataTypes": [{`	array (object), null	
`"format": "isoFid",`	string	✓
`"algorithm": "ecb",`	string, null	
`"keyName": "Key01"`	string, null	
`}],`		
`"numCaptures": 0,`	integer	✓
`"mode": "scan"`	string	✓
`}`		

Properties

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 null. Single or multiple formats can be returned, or no data can be returned in the case where the scan is to be followed by a subsequent matching operation.

default: null

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 null if the biometric data is not encrypted. Available values are described in the encryptionAlgorithms. 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. default: null
dataTypes/keyName Specifies the name of the key that is used to encrypt the biometric data. This property is null if the biometric data is not encrypted. The detailed key information is available through the KeyManagement.GetKeyDetail. default: null
numCaptures This property indicates the number of times to attempt capture of the biometric data from the subject. If this is zero, then the device determines how many attempts will be made. The maximum number of captures possible is indicated by the maxCapture capability. 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 2.0)	Type	Required
`{`		
`"errorCode": "readFailed",`	string, null	
`"dataRead": [{`	array (object), null	
`"type": {`	object	✓
`"format": "isoFid",`	string	✓
`"algorithm": "ecb",`	string, null	

Payload (version 2.0)	Type	Required
"keyName": "Key01"	string, null	
},		
"data": "1a987D000012Bb"	string	✓
}]		
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

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 null.

default: null

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 null if the biometric data is not encrypted. Available values are described in the encryptionAlgorithms. 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.

default: null

dataRead/type/keyName

Specifies the name of the key that is used to encrypt the biometric data. This property is null if the biometric data is not encrypted. The detailed key information is available through the KeyManagement.GetKeyDetail.

default: null

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

18.2.3 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.Clear is called. If template data has been previously imported using a call to Biometric.Import, then it is overwritten. This data is not persistent across power fails.

Command Message

Payload (version 2.0)	Type	Required
`{`		
`"templates": [{`	array (object)	✓
`"type": {`	object	✓
`"format": "isoFid",`	string	✓
`"algorithm": "ecb",`	string, null	
`"keyName": "Key01"`	string, null	
`},`		
`"data": "1a987D000012Bb"`	string	✓
`}]`		
`}`		

Properties

templates

Array of template data to be imported in the device.

Property value constraints:

```
minItems: 1
```

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 null if the biometric data is not encrypted. Available values are described in the encryptionAlgorithms. 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. default: null
templates/type/keyName Specifies the name of the key that is used to encrypt the biometric data. This property is null if the biometric data is not encrypted. The detailed key information is available through the KeyManagement.GetKeyDetail. default: null

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 2.0)	Type	Required
{		
"errorCode": "invalidData",	string, null	
"templates": {	object, null	
"*id1*": {	object	
"format": "isoFid",	string	✓
"algorithm": "ecb",	string, null	
"keyName": "Key01"	string, null	
},		
"*id2*": See *templates/id1* properties	object	
}		
}		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null

Properties
templates A list of the biometric template data that were successfully imported. If there are no template data imported, this property can be null. default: null
templates/id1 (example name) A unique identifier of the biometric template data. 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 null if the biometric data is not encrypted. Available values are described in the encryptionAlgorithms. 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. default: null
templates/id1/keyName Specifies the name of the key that is used to encrypt the biometric data. This property is null if the biometric data is not encrypted. The detailed key information is available through the KeyManagement.GetKeyDetail. default: null

Event Messages

None

18.2.4 Biometric.Match

This command returns the result of a comparison between data that has been scanned using the Biometric.Read and 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 2.0)	Type	Required
`{`		
`"compareMode": "verify",`	string	✓
`"identifier": "id1",`	string, null	
`"maximum": 0,`	integer, null	
`"threshold": 80`	integer	✓
`}`		

Properties

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 null. This property corresponds to the list of template identifiers returned by the Biometric.GetStorageInfo command. Property value constraints: `pattern: ^id[0-9A-Za-z]+$` default: null
maximum Specifies the maximum number of matches to return. In the case where *compareMode* is verify this property can be null. Property value constraints: `minimum: 0` default: null

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 2.0)	Type	Required
{		
"errorCode": "noImportedData",	string, null	
"candidates": {	object, null	
"*id1*": {	object	
"confidenceLevel": 0,	integer	
"templateData": "dGVtcGxhdGUgZGF0YQ=="	string, null	
},		
"*id2*": See *candidates/id1* properties	object	
}		
}		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null

Properties
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 null 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. default: null
candidates/id1 (example name) A unique identifier of the biometric template data. 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`
default: 0

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 is null if no additional comparison data is returned.
Property value constraints:
`pattern: ^[A-Za-z0-9+/]+={0,2}?$`
`format: base64`
default: null

Event Messages

None

18.2.5 Biometric.SetMatch

This command is used for devices which need to know the match criteria data for the Biometric.Match before 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 2.0)	Type	Required
{		
"compareMode": "verify",	string	✓
"identifier": "id1",	string, null	
"maximum": 0,	integer, null	
"threshold": 80	integer	✓
}		

Properties

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 null. This property corresponds to the list of template identifiers returned by the Biometric.GetStorageInfo command.

Property value constraints:

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

default: null

maximum

Specifies the maximum number of matches to return. In the case where *compareMode* is verify this property can be null.

Property value constraints:

```
minimum: 0
```

default: null

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 2.0)	Type	Required
{		
"errorCode": "invalidIdentifier"	string, null	
}		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null

Event Messages

None

18.2.6 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 2.0)	Type	Required
{		
"clearData": "scannedData"	string, null	
}		

Properties

clearData

This property indicates the type of data to be or which has been cleared from storage. If this property is null, then all stored data will be or 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.

default: null

Completion Message

Payload (version 2.0)
This message does not define any properties.

Event Messages

None

18.2.7 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 2.0)	Type	Required
{		
"clearData": "scannedData"	string, null	
}		

Properties

clearData

This property indicates the type of data to be or which has been cleared from storage. If this property is null, then all stored data will be or 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.

default: null

Completion Message

Payload (version 2.0)
This message does not define any properties.

Event Messages

None

18.2.8 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 2.0)	Type	Required
{		
"persistenceMode": "persist"	string	✓
}		

Properties

persistenceMode

Specifies the data persistence mode. This controls how biometric data that has been captured using the Biometric.Read command will persist. This value itself is persistent. 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 2.0)	Type	Required
{		
"errorCode": "modeNotSupported"	string, null	
}		

Properties

errorCode

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

- `modeNotSupported` - The command failed because a mode was specified which is not supported.

default: null

Event Messages

None

18.3 Event Messages

18.3.1 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.

Event Message

Payload (version 2.0)
`This message does not define any properties.`

18.3.2 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.

Event Message

Payload (version 2.0)
This message does not define any properties.

18.3.3 Biometric.RemoveSubjectEvent

This event is used to notify a client that the subject should be removed from the capture area of the device.

Event Message

Payload (version 2.0)
`This message does not define any properties.`

18.4 Unsolicited Messages

18.4.1 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 2.0)
`This message does not define any properties.`

18.4.2 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 2.0)	Type	Required
`{`		
`"clearData": "scannedData"`	string, null	
`}`		

Properties

clearData

This property indicates the type of data to be or which has been cleared from storage. If this property is null, then all stored data will be or 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.

default: null

18.4.3 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 2.0)
`This message does not define any properties.`

19. 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.

19.1 Command Messages

19.1.1 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 2.0)	Type	Required
{		
"camera": "room",	string	✓
"camData": "Camera 1 Text"	string, null	
}		

Properties

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.

default: null

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "cameraNotSupported",	string, null	
"pictureFile": "Xhdjyedh736ydw7hdi"	string, null	
}		

Properties

errorCode

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

- `cameraNotSupported` - The specified camera is not supported.
- `mediaFull` - The recording media is full.
- `cameraInoperable` - The specified camera is inoperable.

default: null

pictureFile

The base64 encoded data representing the picture.

Property value constraints:

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

default: null

Event Messages

- Camera.InvalidDataEvent

19.1.2 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 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)
`This message does not define any properties.`

Event Messages

None

19.2 Event Messages

19.2.1 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 2.0)
This message does not define any properties.

19.3 Unsolicited Messages

19.3.1 Camera.MediaThresholdEvent

This event is used to specify that the state of the recording media reached a threshold.

Unsolicited Message

Payload (version 2.0)	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.

20. 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.

20.1 Command Messages

20.1.1 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 2.0)	Type	Required
{		
"cardReader": {	object, null	
"position": "left",	string	✓
"flashRate": "off",	string, null	
"color": "red",	string, null	
"direction": "entry"	string, null	
},		
"pinPad": See cardReader properties	object, null	
"notesDispenser": See cardReader properties	object, null	
"coinDispenser": See cardReader properties	object, null	
"receiptPrinter": See cardReader properties	object, null	
"passbookPrinter": See cardReader properties	object, null	
"envelopeDepository": See cardReader properties	object, null	
"checkUnit": See cardReader properties	object, null	
"billAcceptor": See cardReader properties	object, null	
"envelopeDispenser": See cardReader properties	object, null	
"documentPrinter": See cardReader properties	object, null	
"coinAcceptor": See cardReader properties	object, null	
"scanner": See cardReader properties	object, null	
"contactless": See cardReader properties	object, null	
"cardReader2": See cardReader properties	object, null	
"notesDispenser2": See cardReader properties	object, null	
"billAcceptor2": See cardReader properties	object, null	
"statusGood": See cardReader properties	object, null	
"statusWarning": See cardReader properties	object, null	
"statusBad": See cardReader properties	object, null	
"statusSupervisor": See cardReader properties	object, null	
"statusInService": See cardReader properties	object, null	
"fasciaLight": See cardReader properties	object, null	
"*vendorSpecificLight*": See *cardReader* properties	object, null	
}		

Properties
cardReader Card Reader Light. This property is null if not applicable. default: null
cardReader/position The light position. Can be used for devices which have multiple input and output positions. This may be 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. • `default` - The default position.
cardReader/flashRate The light flash rate. This may be null in Common.StatusChangedEvent if unchanged, otherwise 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). default: null
cardReader/color The light color. This may be null in Common.StatusChangedEvent if unchanged, otherwise 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. default: null
cardReader/direction The light direction, The value can be null if not required. One of the following values: • `entry` - The light is indicating entry. • `exit` - The light is indicating exit. default: null
pinPad Pin Pad Light. This property is null if not applicable. default: null
notesDispenser Notes Dispenser Light. This property is null if not applicable. default: null
coinDispenser Coin Dispenser Light. This property is null if not applicable. default: null

Properties
receiptPrinter Receipt Printer Light. This property is null if not applicable. default: null
passbookPrinter Passbook Printer Light. This property is null if not applicable. default: null
envelopeDepository Envelope Depository Light. This property is null if not applicable. default: null
checkUnit Check Unit Light. This property is null if not applicable. default: null
billAcceptor Bill Acceptor Light. This property is null if not applicable. default: null
envelopeDispenser Envelope Dispenser Light. This property is null if not applicable. default: null
documentPrinter Document Printer Light. This property is null if not applicable. default: null
coinAcceptor Coin Acceptor Light. This property is null if not applicable. default: null
scanner Scanner Light. This property is null if not applicable. default: null
contactless Contactless Reader Light. This property is null if not applicable. default: null
cardReader2 Card Reader 2 Light. This property is null if not applicable. default: null
notesDispenser2 Notes Dispenser 2 Light. This property is null if not applicable. default: null
billAcceptor2 Bill Acceptor 2 Light. This property is null if not applicable. default: null
statusGood Status Indicator light - Good. This property is null if not applicable. default: null
statusWarning Status Indicator light - Warning. This property is null if not applicable. default: null

Properties
statusBad Status Indicator light - Bad. This property is null if not applicable. default: null
statusSupervisor Status Indicator light - Supervisor. This property is null if not applicable. default: null
statusInService Status Indicator light - In Service. This property is null if not applicable. default: null
fasciaLight Fascia Light. This property is null if not applicable. default: null
vendorSpecificLight (example name) Additional vendor specific lights. default: null

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "invalidLight"`	string, null	
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null

Event Messages

None

21. 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.

When status information of the auxiliaries unit is changed, the Common.StatusChangedEvent is posted.

21.1 Command Messages

21.1.1 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 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
{		
"mode": "specific",	string	✓
"startTime": {	object	✓
"year": 1601,	integer, null	
"month": 1,	integer, null	
"dayOfWeek": "Saturday",	string, null	
"day": 1,	integer, null	
"hour": 0,	integer, null	
"minute": 0	integer, null	
}		
}		

Properties

mode

Specifies the current or desired auto start-up control mode configured. The following values are possible:

- `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.

Property value constraints:

```
minProperties: 2
```

startTime/year

Specifies the year. This property is null if it is not relevant to the *mode*.

Property value constraints:

```
minimum: 1601
maximum: 30827
```

default: null

Properties

startTime/month
Specifies the month. This property is null if it is not relevant to the *mode*.
Property value constraints:
`minimum: 1`
`maximum: 12`
default: null

startTime/dayOfWeek Specifies the day of the week. This property is null if it is not 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. default: null

startTime/day
Specifies the day. This property is null if it is not relevant to the *mode*.
Property value constraints:
`minimum: 1`
`maximum: 31`
default: null

startTime/hour
Specifies the hour. This property is null if it is not relevant to the *mode*.
Property value constraints:
`minimum: 0`
`maximum: 23`
default: null

startTime/minute
Specifies the minute. This property is null if it is not relevant to the *mode*.
Property value constraints:
`minimum: 0`
`maximum: 59`
default: null

Event Messages

None

21.1.2 Auxiliaries.ClearAutoStartupTime

This command is used to clear the time at which the machine will automatically start.

Command Message

Payload (version 2.0)
This message does not define any properties.

Completion Message

Payload (version 2.0)
This message does not define any properties.

Event Messages

None

21.1.3 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 2.0)	Type	Required
{		
"operatorSwitch": "register",	string, null	
"tamperSensor": "register",	string, null	
"internalTamperSensor": "register",	string, null	
"seismicSensor": "register",	string, null	
"heatSensor": "register",	string, null	
"proximitySensor": "register",	string, null	
"ambientLightSensor": "register",	string, null	
"enhancedAudioSensor": "register",	string, null	
"bootSwitchSensor": "register",	string, null	
"consumerDisplaySensor": "register",	string, null	
"operatorCallButtonSensor": "register",	string, null	
"handsetSensor": "register",	string, null	
"headsetMicrophoneSensor": "register",	string, null	
"fasciaMicrophoneSensor": "register",	string, null	
"cabinetDoor": "register",	string, null	
"safeDoor": "register",	string, null	
"vandalShield": "register",	string, null	
"cabinetFront": "register",	string, null	
"cabinetRear": "register",	string, null	
"cabinetRight": "register",	string, null	
"cabinetLeft": "register",	string, null	
"openCloseIndicator": "register",	string, null	
"audioIndicator": "register",	string, null	
"heatingIndicator": "register",	string, null	
"consumerDisplayBacklight": "register",	string, null	
"signageDisplay": "register",	string, null	
"volume": "register",	string, null	
"ups": "register",	string, null	
"audibleAlarm": "register",	string, null	
"enhancedAudioControl": "register",	string, null	
"enhancedMicrophoneControl": "register",	string, null	

Payload (version 2.0)	Type	Required
"microphoneVolume": "register",	string, null	
"exampleProperty1": "register",	string, null	
"exampleProperty2": See exampleProperty1	string, null	
}		

Properties

operatorSwitch

Specifies whether the Operator Switch should report whenever the switch changes the operating mode:

- `register` - Report when this sensor is triggered.
- `deregister` - Do not report when this sensor is triggered.

This property is null if not applicable.

default: null

tamperSensor

Specifies whether the Tamper Sensor should report whenever someone tampers with the terminal. See operatorSwitch for the possible values. This property is null if not applicable.

default: null

internalTamperSensor

Specifies whether the Internal Tamper Sensor should report whenever someone tampers with the internal alarm. See operatorSwitch for the possible values. This property is null if not applicable.

default: null

seismicSensor

Specifies whether the Seismic Sensor should report whenever any seismic activity is detected. See operatorSwitch for the possible values. This property is null if not applicable.

default: null

heatSensor

Specifies whether the Heat Sensor should report whenever any excessive heat is detected. See operatorSwitch for the possible values. This property is null if not applicable.

default: null

proximitySensor

Specifies whether the Proximity Sensor should report whenever any movement is detected close to the terminal. See operatorSwitch for the possible values. This property is null if not applicable.

default: null

ambientLightSensor

Specifies whether the Ambient Light Sensor should report whenever it detects changes in the ambient light. See operatorSwitch for the possible values. This property is null if not applicable.

default: null

enhancedAudioSensor

Specifies whether the Audio Jack should report whenever it detects changes in the audio jack. See operatorSwitch for the possible values. This property is null if not applicable.

default: null

bootSwitchSensor

Specifies whether the Boot Switch should report whenever the delayed effect boot switch is used. See operatorSwitch for the possible values. This property is null if not applicable.

default: null

consumerDisplaySensor

Specifies whether the Consumer Display Sensor should report whenever it detects changes to the consumer display. See operatorSwitch for the possible values. This property is null if not applicable.

default: null

Properties
operatorCallButtonSensor Specifies whether the Operator Call Button should report whenever the Operator Call Button is pressed or released. See operatorSwitch for the possible values. This property is null if not applicable. default: null
handsetSensor Specifies whether the Handset Sensor should report whenever it detects changes of its status. See operatorSwitch for the possible values. This property is null if not applicable. default: null
headsetMicrophoneSensor Specifies whether the Microphone Jack should report whenever it detects changes in the microphone jack. See operatorSwitch for the possible values. This property is null if not applicable. default: null
fasciaMicrophoneSensor Specifies whether the Fascia Microphone should report whenever it detects changes in the microphone state. See operatorSwitch for the possible values. This property is null if not applicable. default: null
cabinetDoor Specifies whether the Cabinet Doors should report whenever the doors are opened, closed, bolted or locked. See operatorSwitch for the possible values. This property is null if not applicable. default: null
safeDoor Specifies whether the Safe Doors should report whenever the doors are opened, closed, bolted or locked. See operatorSwitch for the possible values. This property is null if not applicable. default: null
vandalShield Specifies whether the Vandal Shield should report whenever the shield changed position. See operatorSwitch for the possible values. This property is null if not applicable. default: null
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. This property is null if not applicable. default: null
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. This property is null if not applicable. default: null
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. This property is null if not applicable. default: null
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. This property is null if not applicable. default: null
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. This property is null if not applicable. default: null

Properties
audioIndicator Specifies whether the Audio Indicator should report whenever it is turned on or turned off. See operatorSwitch for the possible values. This property is null if not applicable. default: null
heatingIndicator Specifies whether the Heating device should report whenever it is turned on or turned off. See operatorSwitch for the possible values. This property is null if not applicable. default: null
consumerDisplayBacklight Specifies whether the Consumer Display Backlight should report whenever it is turned on or turned off. See operatorSwitch for the possible values. This property is null if not applicable. default: null
signageDisplay Specifies whether the Signage Display should report whenever it is turned on or turned off. See operatorSwitch for the possible values. This property is null if not applicable. default: null
volume Specifies whether the Volume Control device should report whenever it is changed. See operatorSwitch for the possible values. This property is null if not applicable. default: null
ups Specifies whether the UPS device should report whenever it is changed. See operatorSwitch for the possible values. This property is null if not applicable. default: null
audibleAlarm Specifies whether the Audible Alarm device should report whenever it is changed. See operatorSwitch for the possible values. This property is null if not applicable. default: null
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. This property is null if not applicable. default: null
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. This property is null if not applicable. default: null
microphoneVolume Specifies whether the Microphone Volume Control device should report whenever it is changed. See operatorSwitch for the possible values. This property is null if not applicable. default: null
exampleProperty1 (example name) Specifies whether the vendor dependent sensors should report whenever they change status. See operatorSwitch for the possible values. This property is null if not applicable. default: null

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "invalidAuxiliary"	string, null	
}		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

Event Messages

None

21.1.4 Auxiliaries.SetAuxiliaries

This command is used to set or clear one or more device auxiliaries.

Command Message

Payload (version 2.0)	Type	Required
{		
"cabinetDoors": "bolt",	string, null	
"safeDoor": "bolt",	string, null	
"vandalShield": "closed",	string, null	
"frontCabinetDoors": "bolt",	string, null	
"rearCabinetDoors": "bolt",	string, null	
"leftCabinetDoors": "bolt",	string, null	
"rightCabinetDoors": "bolt",	string, null	
"openClose": "closed",	string, null	
"audio": {	object, null	
"rate": "on",	string	✓
"signal": "keypress"	string	✓
},		
"heating": "on",	string, null	
"consumerDisplayBackLight": "on",	string, null	
"signageDisplay": "on",	string, null	
"volume": 1,	integer, null	
"ups": "engage",	string, null	
"audibleAlarm": "off",	string, null	
"enhancedAudioControl": "publicAudioManual",	string, null	
"enhancedMicrophoneControl": "publicAudioManual",	string, null	
"microphoneVolume": 1	integer, null	
}		

Properties

cabinetDoors

Specifies whether all the Cabinet Doors should be bolted or unbolted as one of the following values:

- `bolt` - All Cabinet Doors are bolted.
- `unbolt` - All Cabinet Doors are unbolted.

This property is null if not applicable.

default: null

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.

This property is null if not applicable.

default: null

Properties
vandalShield Specifies whether the Vandal Shield should change position as one of the following values: • `closed` - Close the Vandal Shield. • `open` - Open the Vandal Shield. • `service` - Position the Vandal Shield in the service position. • `keyboard` - Position the Vandal Shield to permit access to the keyboard. This property is null if not applicable. default: null
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. This property is null if not applicable. default: null
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. This property is null if not applicable. default: null
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. This property is null if not applicable. default: null
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. This property is null if not applicable. default: null
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. This property is null if not applicable. default: null
audio Specifies whether the Audio Indicator should be turned on or off, if available. This property is null if not applicable. default: null
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.

Properties
audio/signal Specifies the Audio sound as one of the following values: • `keypress` - Sound a key click signal. • `exclamation` - Sound an exclamation signal. • `warning` - Sound a warning signal. • `error` - Sound an error signal. • `critical` - Sound a critical error signal.
heating Specifies whether the Internal Heating device should be turned on or off as one of the following values: • `off` - The Internal Heating device is turned off. • `on` - The Internal Heating device is turned on. This property is null if not applicable. default: null
consumerDisplayBackLight 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. This property is null if not applicable. default: null
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. This property is null if not applicable. default: null

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. This property is null if not applicable.
Property value constraints:
`minimum: 1`
`maximum: 1000`
default: null

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. This property is null if not applicable. default: null
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. This property is null if not applicable. default: null

Properties

enhancedAudioControl

Specifies whether the state of the Enhanced Audio Controller should be changed as one of the following values:

- `publicAudioManual` - Set the Enhanced Audio Controller to manual mode, public

state (i.e. audio will be played through speakers only).

- `publicAudioAuto` - Set the Enhanced Audio Controller to auto mode, public state

(i.e. audio will be played through speakers). When a Privacy Device is activated (headset connected/handset off-hook), the device will go to the private state.

- `publicAudioSemiAuto` - Set the Enhanced Audio Controller to semi-auto mode, public

state (i.e. audio will be played through speakers). When a Privacy Device is activated, the device will go to the private state.

- `privateAudioManual` - Set the Enhanced Audio Controller to manual mode, private

state (i.e. audio will be played only through a connected Privacy Device). In private mode, no audio is transmitted through the speakers.

- `privateAudioAuto` - Set the Enhanced Audio Controller to auto mode, private state

(i.e. audio will be played only through an activated Privacy Device). In private mode, no audio is transmitted through the speakers. When a Privacy Device is deactivated (headset disconnected/handset on-hook), the device will go to the public state.

- `privateAudioSemiAuto` - Set the Enhanced Audio Controller to semi-auto mode,

private state (i.e. audio will be played only through an activated Privacy Device). In private mode, no audio is transmitted through the speakers. When a Privacy Device is deactivated, the device will remain in the private state.

This property is null if not applicable.

default: null

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.

This property is null if not applicable.

default: null

Properties

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. This property is null if not applicable.
Property value constraints:
`minimum: 1`
`maximum: 1000`
default: null

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "invalidAuxiliary"`	string, null	
`}`		

Properties
errorCode Specifies the error code if applicable, otherwise null. 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. default: null

Event Messages

None

21.1.5 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 2.0)	Type	Required
{		
"mode": "specific",	string	✓
"startTime": {	object	✓
"year": 1601,	integer, null	
"month": 1,	integer, null	
"dayOfWeek": "Saturday",	string, null	
"day": 1,	integer, null	
"hour": 0,	integer, null	
"minute": 0	integer, null	
}		
}		

Properties

mode

Specifies the current or desired auto start-up control mode configured. The following values are possible:

- `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.

Property value constraints:

```
minProperties: 2
```

startTime/year

Specifies the year. This property is null if it is not relevant to the *mode*.

Property value constraints:

```
minimum: 1601
maximum: 30827
```

default: null

startTime/month

Specifies the month. This property is null if it is not relevant to the *mode*.

Property value constraints:

```
minimum: 1
maximum: 12
```

default: null

Properties
startTime/dayOfWeek Specifies the day of the week. This property is null if it is not 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. default: null

startTime/day
Specifies the day. This property is null if it is not relevant to the *mode*.
Property value constraints:
`minimum: 1`
`maximum: 31`
default: null

startTime/hour
Specifies the hour. This property is null if it is not relevant to the *mode*.
Property value constraints:
`minimum: 0`
`maximum: 23`
default: null

startTime/minute
Specifies the minute. This property is null if it is not relevant to the *mode*.
Property value constraints:
`minimum: 0`
`maximum: 59`
default: null

Completion Message

Payload (version 2.0)
`This message does not define any properties.`

Event Messages

None

22. 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.

22.1 General Information

22.1.1 Transaction Flows

The following sections describe how various scenarios are handled using XFS4IoT Storage.

Replenishment of a Cash Handling device

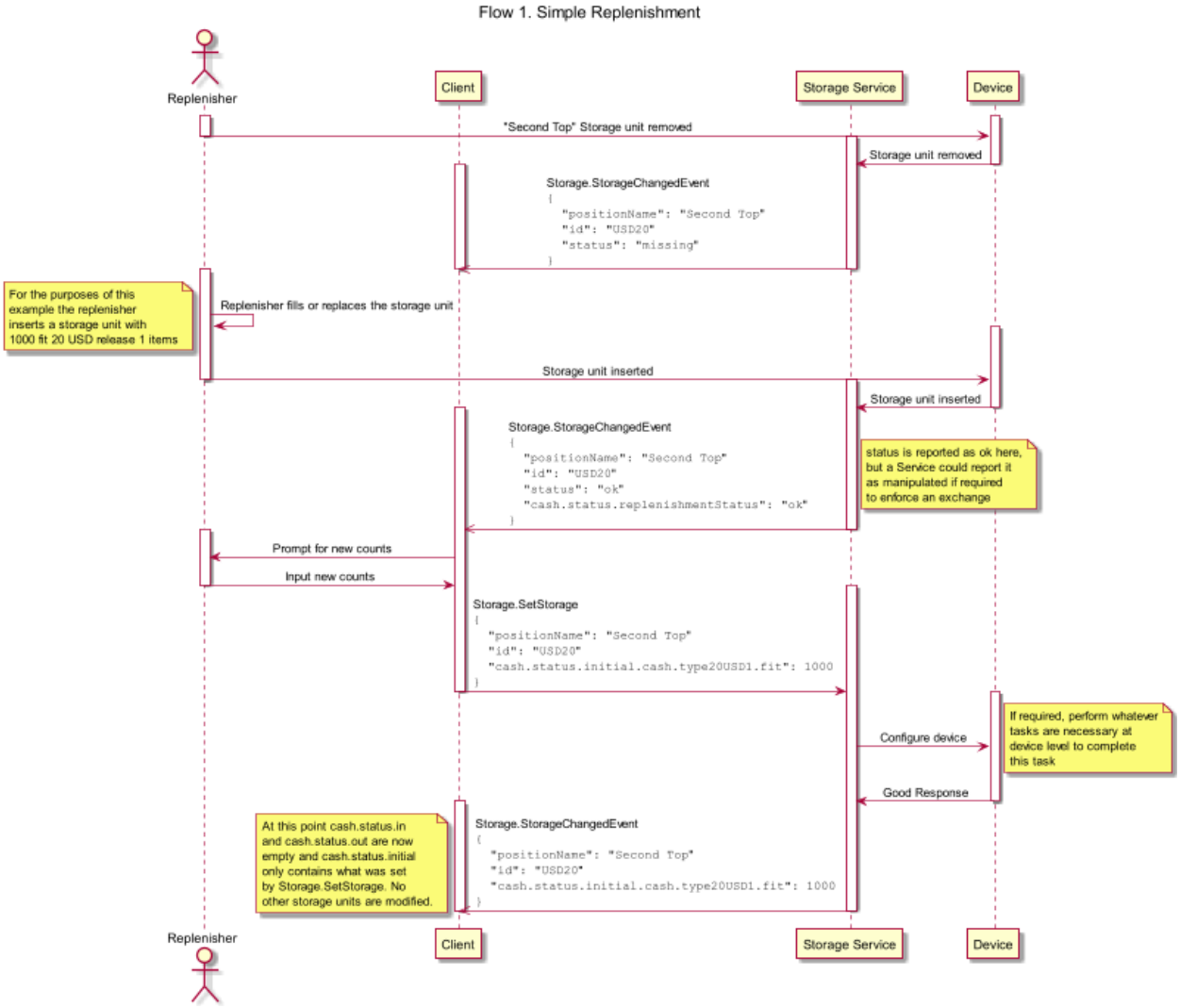
Manual Cash Replenishment in XFS4IoT is performed using the Storage.SetStorage command.

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 properties can be set.

Flow 1 - No Exchange

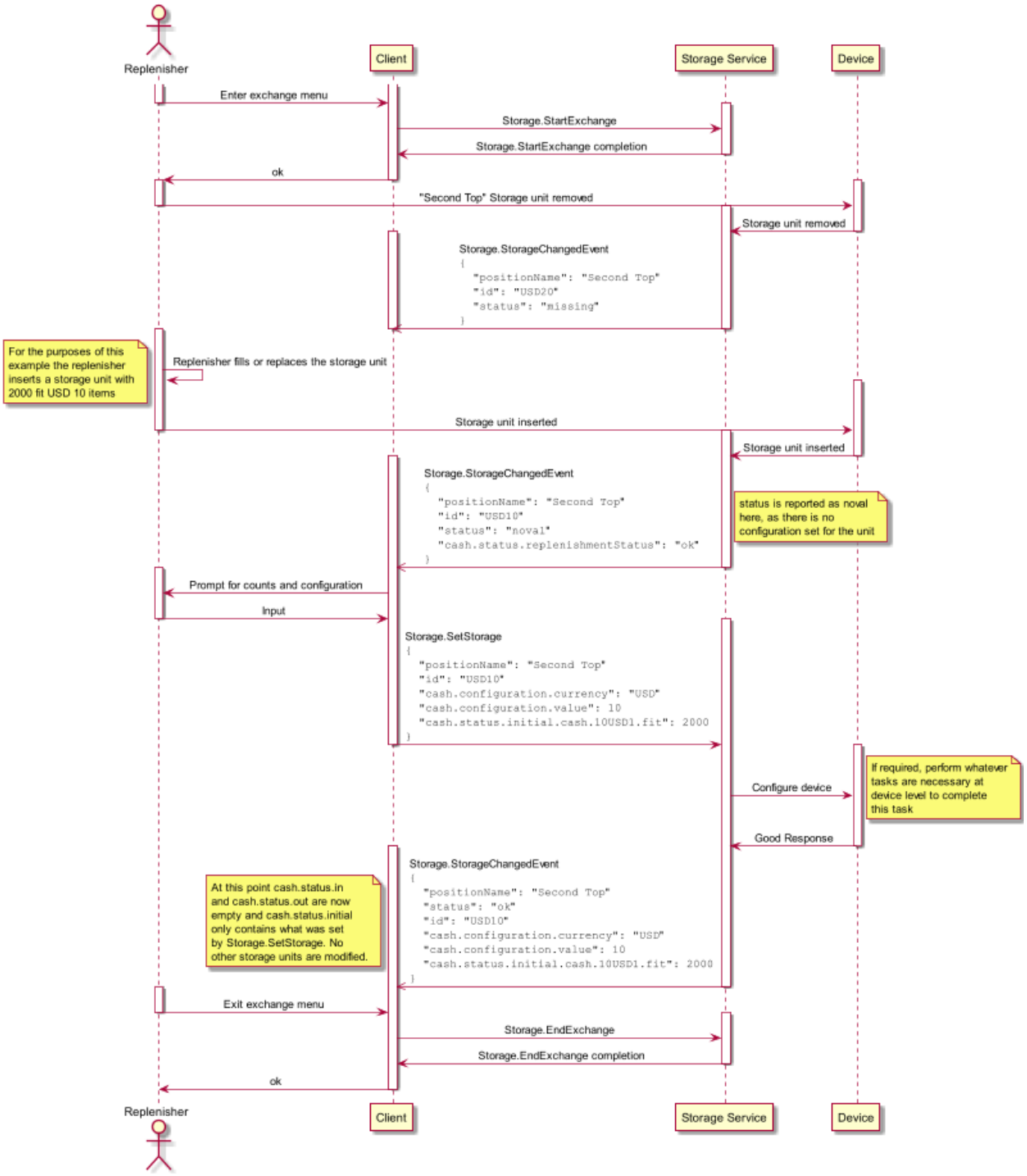
In this flow, one or more storage units are replenished and as only counts have changed, an exchange session is not required. In this scenario the replenisher removes a storage unit and replaces it with one which contains 1000 USD 20 notes.



Flow 2 - With Exchange

In this flow, 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



22.2 Command Messages

22.2.1 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 2.0)
This message does not define any properties.

Completion Message

Payload (version 2.0)	Type	Required
{		
"storage": {	object, null	
"unit1": {	object	
"id": "RC1",	string, null	
"positionName": "Top Right",	string, null	
"capacity": 100,	integer, null	
"status": "ok",	string, null	
"serialNumber": "ABCD1234",	string, null	
"cash": {	object, null	
"capabilities": {	object, null	
"types": {	object, null	
"cashIn": true,	boolean, null	
"cashOut": false,	boolean, null	
"replenishment": false,	boolean, null	
"cashInRetract": false,	boolean, null	
"cashOutRetract": false,	boolean, null	
"reject": false	boolean, null	
},		
"items": {	object, null	
"fit": false,	boolean, null	
"unfit": false,	boolean, null	
"unrecognized": false,	boolean, null	
"counterfeit": false,	boolean, null	
"suspect": false,	boolean, null	
"inked": false,	boolean, null	
"coupon": false,	boolean, null	
"document": false	boolean, null	
},		

Payload (version 2.0)	Type	Required
"hardwareSensors": false,	boolean, null	
"retractAreas": 1,	integer, null	
"retractThresholds": false,	boolean, null	
"cashItems": ["type20USD1", "type50USD1"]	array (string), null	
},		
"configuration": {	object, null	
"types": See storage/unit1/cash/capabilities/types properties	object, null	
"items": See storage/unit1/cash/capabilities/items properties	object, null	
"currency": "USD",	string, null	
"value": 20.00,	number, null	
"highThreshold": 500,	integer, null	
"lowThreshold": 10,	integer, null	
"appLockIn": false,	boolean, null	
"appLockOut": false,	boolean, null	
"cashItems": See storage/unit1/cash/capabilities/cashItems,	array (string), null	
"name": "$10",	string, null	
"maxRetracts": 5	integer, null	
},		
"status": {	object, null	
"index": 4,	integer	✓
"initial": {	object, null	
"unrecognized": 5,	integer, null	
"*type20USD1*": {	object, null	
"fit": 15,	integer, null	
"unfit": 0,	integer, null	
"suspect": 0,	integer, null	
"counterfeit": 0,	integer, null	
"inked": 0	integer, null	
},		
"*type50USD1*": See *storage/unit1/cash/status/initial/type20USD1* properties	object, null	
},		
"out": {	object, null	
"presented": See storage/unit1/cash/status/initial properties	object, null	
"rejected": See storage/unit1/cash/status/initial properties	object, null	
"distributed": See storage/unit1/cash/status/initial properties	object, null	

Payload (version 2.0)	Type	Required
"unknown": See storage/unit1/cash/status/initial properties	object, null	
"stacked": See storage/unit1/cash/status/initial properties	object, null	
"diverted": See storage/unit1/cash/status/initial properties	object, null	
"transport": See storage/unit1/cash/status/initial properties	object, null	
},		
"in": {	object, null	
"retractOperations": 15,	integer, null	
"deposited": See storage/unit1/cash/status/initial properties	object, null	
"retracted": See storage/unit1/cash/status/initial properties	object, null	
"rejected": See storage/unit1/cash/status/initial properties	object, null	
"distributed": See storage/unit1/cash/status/initial properties	object, null	
"transport": See storage/unit1/cash/status/initial properties	object, null	
},		
"accuracy": "accurate",	string, null	
"replenishmentStatus": "ok",	string, null	
"operationStatus": "dispenseInoperative"	string, null	
}		
},		
"card": {	object, null	
"capabilities": {	object, null	
"type": "retain",	string, null	
"hardwareSensors": true	boolean, null	
},		
"configuration": {	object, null	
"cardID": "LoyaltyCard",	string, null	
"threshold": 10	integer, null	
},		
"status": {	object, null	
"initialCount": 0,	integer, null	
"count": 0,	integer, null	
"retainCount": 0,	integer, null	
"replenishmentStatus": "ok"	string, null	
}		
},		
"check": {	object, null	

Payload (version 2.0)	Type	Required
"capabilities": {	object, null	
"types": {	object, null	
"mediaIn": true,	boolean, null	
"retract": false	boolean, null	
},		
"sensors": {	object, null	
"empty": false,	boolean, null	
"high": false,	boolean, null	
"full": false	boolean, null	
}		
},		
"configuration": {	object, null	
"types": See storage/unit1/check/capabilities/types properties	object, null	
"binID": "My check bin",	string, null	
"highThreshold": 500,	integer, null	
"retractHighThreshold": 5	integer, null	
},		
"status": {	object, null	
"index": 4,	integer	✓
"initial": {	object, null	
"mediaInCount": 100,	integer, null	
"count": 150,	integer, null	
"retractOperations": 15	integer, null	
},		
"in": See storage/unit1/check/status/initial properties	object, null	
"replenishmentStatus": "high"	string, null	
}		
}		
},		
"*unit2*": See *storage/unit1* properties	object	
}		
}		

Properties

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

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. May be null if not applicable. Property value constraints: `pattern: ^.{1,5}$` default: null
storage/unit1/positionName Fixed physical name for the position. May be null if not applicable. default: null
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, null means capacity is not applicable. Property value constraints: `minimum: 0` default: null
storage/unit1/status The state of the unit. This property may be null in events if the state did not change, otherwise 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. default: null
storage/unit1/serialNumber The storage unit's serial number if it can be read electronically. May be null if not applicable. default: null
storage/unit1/cash The cash related contents, status and configuration of the unit. May be null if not applicable. default: null
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. This may be null in events if capabilities have not changed. default: null
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. May be null in command data or events if not changed or being changed. default: null
storage/unit1/cash/capabilities/types/cashIn The unit can accept cash items. If *cashOut* is also true then the unit can recycle. May be null in command data or events if not changed or being changed. default: null

Properties
storage/unit1/cash/capabilities/types/cashOut The unit can dispense cash items. If *cashIn* is also true then the unit can recycle. May be null in command data or events if not changed or being changed. default: null
storage/unit1/cash/capabilities/types/replenishment Replenishment container. A storage unit can be refilled from or emptied to a replenishment container. May be null in command data or events if not changed or being changed. default: null
storage/unit1/cash/capabilities/types/cashInRetract Retract unit. Items can be retracted into this unit during Cash In operations. May be null in command data or events if not changed or being changed. default: null
storage/unit1/cash/capabilities/types/cashOutRetract Retract unit. Items can be retracted into this unit during Cash Out operations. May be null in command data or events if not changed or being changed. default: null
storage/unit1/cash/capabilities/types/reject Reject unit. Items can be rejected into this unit. May be null in command data or events if not changed or being changed. default: null
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. May be null in command data if not being changed. May be null in command data or events if not changed or being changed. default: null
storage/unit1/cash/capabilities/items/fit The storage unit can store cash items which are fit for recycling. May be null in command data or events if not changed or being changed. default: null
storage/unit1/cash/capabilities/items/unfit The storage unit can store cash items which are unfit for recycling. May be null in command data or events if not changed or being changed. default: false
storage/unit1/cash/capabilities/items/unrecognized The storage unit can store unrecognized cash items. May be null in command data or events if not changed or being changed. default: null
storage/unit1/cash/capabilities/items/counterfeit The storage unit can store counterfeit cash items. May be null in command data or events if not changed or being changed. default: null
storage/unit1/cash/capabilities/items/suspect The storage unit can store suspect counterfeit cash items. May be null in command data or events if not changed or being changed. default: null

Properties
storage/unit1/cash/capabilities/items/inked The storage unit can store cash items which have been identified as ink stained. May be null in command data or events if not changed or being changed. default: null
storage/unit1/cash/capabilities/items/coupon Storage unit containing coupons or advertising material. May be null in command data or events if not changed or being changed. default: null
storage/unit1/cash/capabilities/items/document Storage unit containing documents. May be null in command data or events if not changed or being changed. default: null
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. May be null in command data or events if not changed or being changed. default: null
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. May be null if items can not be retracted into this storage unit or in events if not changed or being changed. Property value constraints: `minimum: 1` default: null
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. May be null if items can not be retracted into this storage unit or in events if not changed or being changed. default: null
storage/unit1/cash/capabilities/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. May be null in command data or events if not being modified. Property value constraints: `minItems: 1` default: null
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. May be null in command data or events if no configuration is to be or has been changed. default: null
storage/unit1/cash/configuration/currency ISO 4217 currency identifier [Ref. cashmanagement-1]. May only be modified in an exchange state if applicable. May be null if the unit is configured to store mixed currencies or non-cash items. Property value constraints: `pattern: ^[A-Z]{3}$` default: null

Properties
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. May be null in command data or events if not being modified. Property value constraints: `minimum: 0` default: null
storage/unit1/cash/configuration/highThreshold If specified, replenishmentStatus is 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 null, high is based on hardware sensors if supported - see hardwareSensors. May be null in command data or events if not being modified. Property value constraints: `minimum: 1` default: null
storage/unit1/cash/configuration/lowThreshold If specified, replenishmentStatus is 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 null, low is based on hardware sensors if supported - see hardwareSensors. May be null in command data or events if not being modified. Property value constraints: `minimum: 1` default: null
storage/unit1/cash/configuration/appLockIn If true, items cannot be accepted into the storage unit in Cash In operations. May be null in command data or events if not being modified. default: null
storage/unit1/cash/configuration/appLockOut If true, items cannot be dispensed from the storage unit in Cash Out operations. May be null in command data or events if not being modified. default: null
storage/unit1/cash/configuration/name Application configured name of the unit. May be null in command data or events if not being modified. default: null
storage/unit1/cash/configuration/maxRetracts If specified, this is the number of retract operations allowed into the unit. Only applies to retract units. If retractOperations equals this number, then no further retracts are allowed into this storage unit. If null in output, the maximum number is not limited by counts. May be null in command data or events if not being modified. Property value constraints: `minimum: 1` default: null
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. May be null in events if not changing. default: null

Properties
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`
storage/unit1/cash/status/initial The cash related items which were in the storage unit at the last replenishment. default: null
storage/unit1/cash/status/initial/unrecognized Count of unrecognized items handled by the cash interface. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/cash/status/initial/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification. default: null
storage/unit1/cash/status/initial/type20USD1/fit Count of genuine cash items which are fit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/cash/status/initial/type20USD1/unfit Count of genuine cash items which are unfit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/cash/status/initial/type20USD1/suspect Count of suspected counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/cash/status/initial/type20USD1/counterfeit Count of counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/cash/status/initial/type20USD1/inked Count of cash items which have been identified as ink stained. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null

Properties
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. May be null if items have not or can not be moved from the storage unit by cash commands. default: null
storage/unit1/cash/status/out/presented The items dispensed from this storage unit which are or were customer accessible. Will be null if no items were presented. default: null
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. Will be null if no items were rejected. default: null
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. Will be null if no items were distributed. default: null
storage/unit1/cash/status/out/unknown The items dispensed from this storage unit which moved to an unknown position. Will be null if no items were unknown. default: null
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. Will be null if no items were stacked. default: null
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. Will be null if no items were diverted. default: null
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. Will be null if no items apply. default: null

Properties
storage/unit1/cash/status/in List of items inserted in this storage unit by cash commands from another source since the last replenishment of this unit. 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. May be null if items have not or can not be moved into the storage unit by cash commands. default: null
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. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
storage/unit1/cash/status/in/deposited The items deposited in the storage unit during a Cash In transaction. Can be null, if all values are 0. default: null
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*. Can be null, if all values are 0. default: null
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. Can be null, if all values are 0. default: null
storage/unit1/cash/status/in/distributed The items deposited in this storage unit originating from another storage unit but not rejected. Can be null, if all values are 0. default: null
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. Can be null, if all values are 0. default: null

Properties
storage/unit1/cash/status/accuracy Describes the accuracy of the counts reported by *out* and *in*. If null in Storage.GetStorage, the hardware is not capable of determining the accuracy, otherwise the following values are possible: • `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. default: null
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. May be null in events if not changing, otherwise 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. default: null
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 null in Storage.GetStorage, *status* and *replenishmentStatus* apply to both cash out and cash in operations. default: null
storage/unit1/card The card related contents, status and configuration of the unit. May be null if not applicable. default: null
storage/unit1/card/capabilities Indicates the card storage unit capabilities. This property can be null if a change is being reported using Storage.StorageChangedEvent or Storage.StorageThresholdEvent.

Properties
storage/unit1/card/capabilities/type The type of card storage. This property may be null in events if the type did not change, otherwise will be one of the following values: • `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. default: null
storage/unit1/card/capabilities/hardwareSensors Indicates whether the storage unit has hardware sensors that can detect threshold states. This property may be null in events if it did not change. default: null
storage/unit1/card/configuration Indicates the card storage unit configuration. This property can be null if the storage unit is being set using Storage.SetStorage, or a change is being reported using Storage.StorageChangedEvent or Storage.StorageThresholdEvent.
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 storage units and may be null in events if it did not change. default: null
storage/unit1/card/configuration/threshold If the threshold value is non zero, hardware sensors in the storage unit do not trigger Storage.StorageThresholdEvent events. This property may be null in events if it did not change. 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` default: null
storage/unit1/card/status Indicates the card storage unit status. This property can be null if a change is being reported using Storage.StorageChangedEvent or Storage.StorageThresholdEvent.
storage/unit1/card/status/initialCount The initial number of cards in the storage unit. This is only applicable to dispense type storage units. This property may be null in events if it did not change. This value is persistent. Property value constraints: `minimum: 0` default: null

Properties
storage/unit1/card/status/count The number of cards in the storage unit. If the storage unit type is dispense: • This count also includes a card dispensed from the storage unit which has not been moved to either the exit position or a dispense type storage unit. • This count is decremented when a card from the card storage unit is moved to the exit position or retained. If this value reaches zero it will not decrement further but will remain at zero. If the storage unit type is retain: • The count is incremented when a card is moved into the storage unit. If the storage unit type is park: • The count will increment when a card is moved into the storage module and decremented when a card is moved out of the storage module. This value is persistent and may be null in events if it did not change. Property value constraints: `minimum: 0` default: null
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 and may be null in events if it did not change. Property value constraints: `minimum: 0` default: null
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 null. The property may also be null in events if it did not change. 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. default: null
storage/unit1/check The check related contents, status and configuration of the unit. May be null if not applicable. default: null
storage/unit1/check/capabilities Indicates what the storage unit is capable of - this includes information which is a combination of that reported in WFS_INF_IPM_MEDIA_BIN_INFO and WFS_INF_IPM_MEDIA_BIN_CAPABILITIES in XFS 3.x. May be null in events if not changed. default: null
storage/unit1/check/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. May be null in command data and events if not changing. default: null

Properties
storage/unit1/check/capabilities/types/mediaIn The unit can accept items during Media In transactions. May be null in command data and events if not changing. default: null
storage/unit1/check/capabilities/types/retract Retract unit. Items can be retracted into this unit using Check.RetractMedia. May be null in command data and events if not changing. default: null
storage/unit1/check/capabilities/sensors The types of sensor the unit has. May be null in command data and events if not changing. default: null
storage/unit1/check/capabilities/sensors/empty The unit contains a hardware sensor which reports when the unit is empty. May be null in command data and events if not changing. default: null
storage/unit1/check/capabilities/sensors/high The unit contains a hardware sensor which reports when the unit is nearly full. May be null in command data and events if not changing. default: null
storage/unit1/check/capabilities/sensors/full The unit contains a hardware sensor which reports when the unit is full. May be null in command data and events if not changing. default: null
storage/unit1/check/configuration Indicates what the storage unit is configured to do - where applicable the supported options can be derived from capabilities. May be null in command data and events if not being modified.
storage/unit1/check/configuration/binID An application defined Storage Unit Identifier. This may be null in events if not changing. default: null
storage/unit1/check/configuration/highThreshold If specified, replenishmentStatus is set to *high* if the total number of items in the storage unit is greater than this number. May be null in command data and events if not being modified. Property value constraints: `minimum: 1` default: null
storage/unit1/check/configuration/retractHighThreshold If specified and the storage unit is configured as *retract*, replenishmentStatus is set to *high* if the total number of retract operations in the storage unit is greater than this number. May be null in command data and events if not being modified. Property value constraints: `minimum: 0` default: null
storage/unit1/check/status Indicates the storage unit status. May be null in events where status has not changed. default: null

Properties
storage/unit1/check/status/index Assigned by the Service. Will be a unique number which can be used to determine *usBinNumber* in XFS 3.x migration. This can change as storage units are added and removed from the storage collection. Property value constraints: `minimum: 1`
storage/unit1/check/status/initial The check related counts as set at the last replenishment. May be null in events where status has not changed. default: null
storage/unit1/check/status/initial/mediaInCount Count of items added to the storage unit due to Check operations. If the number of items is not counted this is not reported and *retractOperations* is incremented as items are added to the unit. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
storage/unit1/check/status/initial/count Total number of items added to the storage unit due to any operations. If the number of items is not counted this is not reported and *retractOperations* is incremented as items are added to the unit. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
storage/unit1/check/status/initial/retractOperations Total number of operations which resulted in items being retracted to the storage unit. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
storage/unit1/check/status/in The check items added to the unit since the last replenishment. May be null in events where status has not changed. default: null
storage/unit1/check/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. May be null in command data and events if not changing, otherwise 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 a full sensor or counts. • `high` - The storage unit is almost full (either high sensor based or exceeded the highThreshold or retractHighThreshold). • `empty` - The storage unit is empty, or insufficient items in the storage unit are preventing further dispense operations. If the storage unit has the empty sensor, this state is not set by counts. default: null

Event Messages

None

22.2.2 Storage.SetStorage

This command is used to adjust information about the configuration and contents of the device's storage units. Only properties that are to be changed need to be set in the payload of this command; properties that are not meant to change can be null.

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 property 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 2.0)	Type	Required
{		
"storage": {	object	✓
"*unit1*": {	object	
"cash": {	object, null	
"configuration": {	object, null	
"types": {	object, null	
"cashIn": true,	boolean, null	
"cashOut": false,	boolean, null	
"replenishment": false,	boolean, null	
"cashInRetract": false,	boolean, null	
"cashOutRetract": false,	boolean, null	
"reject": false	boolean, null	
},		
"items": {	object, null	
"fit": false,	boolean, null	
"unfit": false,	boolean, null	
"unrecognized": false,	boolean, null	
"counterfeit": false,	boolean, null	
"suspect": false,	boolean, null	
"inked": false,	boolean, null	
"coupon": false,	boolean, null	
"document": false	boolean, null	
},		
"currency": "USD",	string, null	
"value": 20.00,	number, null	
"highThreshold": 500,	integer, null	
"lowThreshold": 10,	integer, null	
"appLockIn": false,	boolean, null	

Payload (version 2.0)	Type	Required
"appLockOut": false,	boolean, null	
"cashItems": ["type20USD1", "type50USD1"],	array (string), null	
"name": "$10",	string, null	
"maxRetracts": 5	integer, null	
},		
"status": {	object, null	
"initial": {	object, null	
"unrecognized": 5,	integer, null	
"*type20USD1*": {	object, null	
"fit": 15,	integer, null	
"unfit": 0,	integer, null	
"suspect": 0,	integer, null	
"counterfeit": 0,	integer, null	
"inked": 0	integer, null	
},		
"*type50USD1*": See *storage/unit1/cash/status/initial/type20USD1* properties	object, null	
}		
}		
},		
"card": {	object, null	
"configuration": {	object, null	
"cardID": "LoyaltyCard",	string, null	
"threshold": 10	integer, null	
},		
"status": {	object, null	
"initialCount": 0	integer	✓
}		
},		
"check": {	object, null	
"configuration": {	object, null	
"types": {	object, null	
"mediaIn": true,	boolean, null	
"retract": false	boolean, null	
},		
"binID": "My check bin",	string, null	
"highThreshold": 500,	integer, null	
"retractHighThreshold": 5	integer, null	
},		
"status": {	object, null	

Payload (version 2.0)	Type	Required
"initial": {	object	✓
"mediaInCount": 100,	integer, null	
"count": 150,	integer, null	
"retractOperations": 15	integer, null	
}		
}		
}		
},		
"*unit2*": See *storage/unit1* properties	object	
}		
}		

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 status and configuration of the unit to be set. May be null if not applicable.
default: null

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.
May be null in command data or events if no configuration is to be or has been changed.
default: null

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. May be null in command data or events if not changed or being changed.
default: null

storage/unit1/cash/configuration/types/cashIn
The unit can accept cash items. If *cashOut* is also true then the unit can recycle. May be null in command data or events if not changed or being changed.
default: null

storage/unit1/cash/configuration/types/cashOut
The unit can dispense cash items. If *cashIn* is also true then the unit can recycle. May be null in command data or events if not changed or being changed.
default: null

storage/unit1/cash/configuration/types/replenishment
Replenishment container. A storage unit can be refilled from or emptied to a replenishment container. May be null in command data or events if not changed or being changed.
default: null

Properties
storage/unit1/cash/configuration/types/cashInRetract Retract unit. Items can be retracted into this unit during Cash In operations. May be null in command data or events if not changed or being changed. default: null
storage/unit1/cash/configuration/types/cashOutRetract Retract unit. Items can be retracted into this unit during Cash Out operations. May be null in command data or events if not changed or being changed. default: null
storage/unit1/cash/configuration/types/reject Reject unit. Items can be rejected into this unit. May be null in command data or events if not changed or being changed. default: null
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. May be null in command data if not being changed. May be null in command data or events if not changed or being changed. default: null
storage/unit1/cash/configuration/items/fit The storage unit can store cash items which are fit for recycling. May be null in command data or events if not changed or being changed. default: null
storage/unit1/cash/configuration/items/unfit The storage unit can store cash items which are unfit for recycling. May be null in command data or events if not changed or being changed. default: false
storage/unit1/cash/configuration/items/unrecognized The storage unit can store unrecognized cash items. May be null in command data or events if not changed or being changed. default: null
storage/unit1/cash/configuration/items/counterfeit The storage unit can store counterfeit cash items. May be null in command data or events if not changed or being changed. default: null
storage/unit1/cash/configuration/items/suspect The storage unit can store suspect counterfeit cash items. May be null in command data or events if not changed or being changed. default: null
storage/unit1/cash/configuration/items/inked The storage unit can store cash items which have been identified as ink stained. May be null in command data or events if not changed or being changed. default: null
storage/unit1/cash/configuration/items/coupon Storage unit containing coupons or advertising material. May be null in command data or events if not changed or being changed. default: null
storage/unit1/cash/configuration/items/document Storage unit containing documents. May be null in command data or events if not changed or being changed. default: null

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 null if the unit is configured to store mixed currencies or non-cash items. Property value constraints: `pattern: ^[A-Z]{3}$` default: null
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. May be null in command data or events if not being modified. Property value constraints: `minimum: 0` default: null
storage/unit1/cash/configuration/highThreshold If specified, replenishmentStatus is 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 null, high is based on hardware sensors if supported - see hardwareSensors. May be null in command data or events if not being modified. Property value constraints: `minimum: 1` default: null
storage/unit1/cash/configuration/lowThreshold If specified, replenishmentStatus is 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 null, low is based on hardware sensors if supported - see hardwareSensors. May be null in command data or events if not being modified. Property value constraints: `minimum: 1` default: null
storage/unit1/cash/configuration/appLockIn If true, items cannot be accepted into the storage unit in Cash In operations. May be null in command data or events if not being modified. default: null
storage/unit1/cash/configuration/appLockOut If true, items cannot be dispensed from the storage unit in Cash Out operations. May be null in command data or events if not being modified. default: null
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. May be null in command data or events if not being modified. Property value constraints: `minItems: 1` default: null
storage/unit1/cash/configuration/name Application configured name of the unit. May be null in command data or events if not being modified. default: null

Properties
storage/unit1/cash/configuration/maxRetracts If specified, this is the number of retract operations allowed into the unit. Only applies to retract units. If retractOperations equals this number, then no further retracts are allowed into this storage unit. If null in output, the maximum number is not limited by counts. May be null in command data or events if not being modified. Property value constraints: `minimum: 1` default: null
storage/unit1/cash/status Set Cash Storage unit properties due to a replenishment or other service action. Only a limited number of properties can be set directly, others may be modified indirectly. May be null if not being modified. default: null
storage/unit1/cash/status/initial The cash related items which are in the storage unit at the last replenishment. If specified, out and in are reset to empty. default: null
storage/unit1/cash/status/initial/unrecognized Count of unrecognized items handled by the cash interface. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/cash/status/initial/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification. default: null
storage/unit1/cash/status/initial/type20USD1/fit Count of genuine cash items which are fit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/cash/status/initial/type20USD1/unfit Count of genuine cash items which are unfit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/cash/status/initial/type20USD1/suspect Count of suspected counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/cash/status/initial/type20USD1/counterfeit Count of counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null

Properties
storage/unit1/cash/status/initial/type20USD1/inked Count of cash items which have been identified as ink stained. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
storage/unit1/card The card related contents and configuration of the unit. May be null if not applicable. default: null
storage/unit1/card/configuration Indicates the card storage unit configuration. This property can be null if the storage unit is being set using Storage.SetStorage, or a change is being reported using Storage.StorageChangedEvent or Storage.StorageThresholdEvent.
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 storage units and may be null in events if it did not change. default: null
storage/unit1/card/configuration/threshold If the threshold value is non zero, hardware sensors in the storage unit do not trigger Storage.StorageThresholdEvent events. This property may be null in events if it did not change. 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` default: null
storage/unit1/card/status Indicates the card storage unit status being set. This property can be null if none of the properties it contains need to be changed.
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`
storage/unit1/check The check related contents, status and configuration of the unit. May be null if not applicable. default: null
storage/unit1/check/configuration Indicates what the storage unit is configured to do - where applicable the supported options can be derived from capabilities. May be null in command data and events if not being modified.
storage/unit1/check/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. May be null in command data and events if not changing. default: null
storage/unit1/check/configuration/types/mediaIn The unit can accept items during Media In transactions. May be null in command data and events if not changing. default: null

Properties
storage/unit1/check/configuration/types/retract Retract unit. Items can be retracted into this unit using Check.RetractMedia. May be null in command data and events if not changing. default: null
storage/unit1/check/configuration/binID An application defined Storage Unit Identifier. This may be null in events if not changing. default: null
storage/unit1/check/configuration/highThreshold If specified, replenishmentStatus is set to *high* if the total number of items in the storage unit is greater than this number. May be null in command data and events if not being modified. Property value constraints: `minimum: 1` default: null
storage/unit1/check/configuration/retractHighThreshold If specified and the storage unit is configured as *retract*, replenishmentStatus is set to *high* if the total number of retract operations in the storage unit is greater than this number. May be null in command data and events if not being modified. Property value constraints: `minimum: 0` default: null
storage/unit1/check/status Set Check Storage unit properties due to a replenishment or other service action. Only a limited number of properties can be set directly, others may be modified indirectly. May be null if not being modified. default: null
storage/unit1/check/status/initial The check related counts as set at the last replenishment.
storage/unit1/check/status/initial/mediaInCount Count of items added to the storage unit due to Check operations. If the number of items is not counted this is not reported and *retractOperations* is incremented as items are added to the unit. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
storage/unit1/check/status/initial/count Total number of items added to the storage unit due to any operations. If the number of items is not counted this is not reported and *retractOperations* is incremented as items are added to the unit. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
storage/unit1/check/status/initial/retractOperations Total number of operations which resulted in items being retracted to the storage unit. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null

Completion Message

Payload (version 2.0)	Type	Required
{		
"errorCode": "invalidUnit"	string, null	
}		

Properties

errorCode
Specifies the error code if applicable, otherwise null. 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.

default: null

Event Messages

- Storage.StorageErrorEvent

22.2.3 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 properties may only be set while in an exchange state, particularly properties which modify the configuration of the storage unit or units. The properties 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 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "storageUnitError"`	string, null	
`}`		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

Event Messages

- Storage.StorageErrorEvent

22.2.4 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 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"errorCode": "storageUnitError"`	string, null	
`}`		

Properties

errorCode

Specifies the error code if applicable, otherwise null. 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.

default: null

Event Messages

- Storage.StorageErrorEvent

22.3 Event Messages

22.3.1 Storage.StorageErrorEvent

This event is generated if there is a problem with a storage unit during the execution of a command.

Event Message

Payload (version 2.0)	Type	Required
{		
"failure": "empty",	string	✓
"unit": {	object	✓
"unit1": {	object	
"id": "RC1",	string, null	
"positionName": "Top Right",	string, null	
"capacity": 100,	integer, null	
"status": "ok",	string, null	
"serialNumber": "ABCD1234",	string, null	
"cash": {	object, null	
"capabilities": {	object, null	
"types": {	object, null	
"cashIn": true,	boolean, null	
"cashOut": false,	boolean, null	
"replenishment": false,	boolean, null	
"cashInRetract": false,	boolean, null	
"cashOutRetract": false,	boolean, null	
"reject": false	boolean, null	
},		
"items": {	object, null	
"fit": false,	boolean, null	
"unfit": false,	boolean, null	
"unrecognized": false,	boolean, null	
"counterfeit": false,	boolean, null	
"suspect": false,	boolean, null	
"inked": false,	boolean, null	
"coupon": false,	boolean, null	
"document": false	boolean, null	
},		
"hardwareSensors": false,	boolean, null	
"retractAreas": 1,	integer, null	
"retractThresholds": false,	boolean, null	
"cashItems": ["type20USD1", "type50USD1"]	array (string), null	

Payload (version 2.0)	Type	Required
},		
"configuration": {	object, null	
"types": See unit/unit1/cash/capabilities/types properties	object, null	
"items": See unit/unit1/cash/capabilities/items properties	object, null	
"currency": "USD",	string, null	
"value": 20.00,	number, null	
"highThreshold": 500,	integer, null	
"lowThreshold": 10,	integer, null	
"appLockIn": false,	boolean, null	
"appLockOut": false,	boolean, null	
"cashItems": See unit/unit1/cash/capabilities/cashItems,	array (string), null	
"name": "$10",	string, null	
"maxRetracts": 5	integer, null	
},		
"status": {	object, null	
"index": 4,	integer	✓
"initial": {	object, null	
"unrecognized": 5,	integer, null	
"*type20USD1*": {	object, null	
"fit": 15,	integer, null	
"unfit": 0,	integer, null	
"suspect": 0,	integer, null	
"counterfeit": 0,	integer, null	
"inked": 0	integer, null	
},		
"*type50USD1*": See *unit/unit1/cash/status/initial/type20USD1* properties	object, null	
},		
"out": {	object, null	
"presented": See unit/unit1/cash/status/initial properties	object, null	
"rejected": See unit/unit1/cash/status/initial properties	object, null	
"distributed": See unit/unit1/cash/status/initial properties	object, null	
"unknown": See unit/unit1/cash/status/initial properties	object, null	
"stacked": See unit/unit1/cash/status/initial properties	object, null	
"diverted": See unit/unit1/cash/status/initial properties	object, null	

Payload (version 2.0)	Type	Required
"transport": See unit/unit1/cash/status/initial properties	object, null	
},		
"in": {	object, null	
"retractOperations": 15,	integer, null	
"deposited": See unit/unit1/cash/status/initial properties	object, null	
"retracted": See unit/unit1/cash/status/initial properties	object, null	
"rejected": See unit/unit1/cash/status/initial properties	object, null	
"distributed": See unit/unit1/cash/status/initial properties	object, null	
"transport": See unit/unit1/cash/status/initial properties	object, null	
},		
"accuracy": "accurate",	string, null	
"replenishmentStatus": "ok",	string, null	
"operationStatus": "dispenseInoperative"	string, null	
}		
},		
"card": {	object, null	
"capabilities": {	object, null	
"type": "retain",	string, null	
"hardwareSensors": true	boolean, null	
},		
"configuration": {	object, null	
"cardID": "LoyaltyCard",	string, null	
"threshold": 10	integer, null	
},		
"status": {	object, null	
"initialCount": 0,	integer, null	
"count": 0,	integer, null	
"retainCount": 0,	integer, null	
"replenishmentStatus": "ok"	string, null	
}		
},		
"check": {	object, null	
"capabilities": {	object, null	
"types": {	object, null	
"mediaIn": true,	boolean, null	
"retract": false	boolean, null	
},		

Payload (version 2.0)	Type	Required
"sensors": {	object, null	
"empty": false,	boolean, null	
"high": false,	boolean, null	
"full": false	boolean, null	
}		
},		
"configuration": {	object, null	
"types": See unit/unit1/check/capabilities/types properties	object, null	
"binID": "My check bin",	string, null	
"highThreshold": 500,	integer, null	
"retractHighThreshold": 5	integer, null	
},		
"status": {	object, null	
"index": 4,	integer	✓
"initial": {	object, null	
"mediaInCount": 100,	integer, null	
"count": 150,	integer, null	
"retractOperations": 15	integer, null	
},		
"in": See unit/unit1/check/status/initial properties	object, null	
"replenishmentStatus": "high"	string, null	
}		
}		
}		
}		
}		

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.
- `feedModuleProblem` - A problem has been detected with the feeding module.
- `physicalLocked` - The storage unit could not be unlocked and remains physically locked.
- `physicalUnlocked` - The storage unit could not be locked and remains physically unlocked.

unit

The storage unit object that caused the problem.

Properties
unit/unit1 (example name) The object contains a single storage unit. Property name constraints: `pattern: ^unit[0-9A-Za-z]+$`
unit/unit1/id An identifier which can be used for cUnitID in CDM/CIM XFS 3.x migration. May be null if not applicable. Property value constraints: `pattern: ^.{1,5}$` default: null
unit/unit1/positionName Fixed physical name for the position. May be null if not applicable. default: null
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, null means capacity is not applicable. Property value constraints: `minimum: 0` default: null
unit/unit1/status The state of the unit. This property may be null in events if the state did not change, otherwise 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. default: null
unit/unit1/serialNumber The storage unit's serial number if it can be read electronically. May be null if not applicable. default: null
unit/unit1/cash The cash related contents, status and configuration of the unit. May be null if not applicable. default: null
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. This may be null in events if capabilities have not changed. default: null
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. May be null in command data or events if not changed or being changed. default: null

Properties
unit/unit1/cash/capabilities/types/cashIn The unit can accept cash items. If *cashOut* is also true then the unit can recycle. May be null in command data or events if not changed or being changed. default: null
unit/unit1/cash/capabilities/types/cashOut The unit can dispense cash items. If *cashIn* is also true then the unit can recycle. May be null in command data or events if not changed or being changed. default: null
unit/unit1/cash/capabilities/types/replenishment Replenishment container. A storage unit can be refilled from or emptied to a replenishment container. May be null in command data or events if not changed or being changed. default: null
unit/unit1/cash/capabilities/types/cashInRetract Retract unit. Items can be retracted into this unit during Cash In operations. May be null in command data or events if not changed or being changed. default: null
unit/unit1/cash/capabilities/types/cashOutRetract Retract unit. Items can be retracted into this unit during Cash Out operations. May be null in command data or events if not changed or being changed. default: null
unit/unit1/cash/capabilities/types/reject Reject unit. Items can be rejected into this unit. May be null in command data or events if not changed or being changed. default: null
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. May be null in command data if not being changed. May be null in command data or events if not changed or being changed. default: null
unit/unit1/cash/capabilities/items/fit The storage unit can store cash items which are fit for recycling. May be null in command data or events if not changed or being changed. default: null
unit/unit1/cash/capabilities/items/unfit The storage unit can store cash items which are unfit for recycling. May be null in command data or events if not changed or being changed. default: false
unit/unit1/cash/capabilities/items/unrecognized The storage unit can store unrecognized cash items. May be null in command data or events if not changed or being changed. default: null
unit/unit1/cash/capabilities/items/counterfeit The storage unit can store counterfeit cash items. May be null in command data or events if not changed or being changed. default: null

Properties
unit/unit1/cash/capabilities/items/suspect The storage unit can store suspect counterfeit cash items. May be null in command data or events if not changed or being changed. default: null
unit/unit1/cash/capabilities/items/inked The storage unit can store cash items which have been identified as ink stained. May be null in command data or events if not changed or being changed. default: null
unit/unit1/cash/capabilities/items/coupon Storage unit containing coupons or advertising material. May be null in command data or events if not changed or being changed. default: null
unit/unit1/cash/capabilities/items/document Storage unit containing documents. May be null in command data or events if not changed or being changed. default: null
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. May be null in command data or events if not changed or being changed. default: null
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. May be null if items can not be retracted into this storage unit or in events if not changed or being changed. Property value constraints: `minimum: 1` default: null
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. May be null if items can not be retracted into this storage unit or in events if not changed or being changed. default: null
unit/unit1/cash/capabilities/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. May be null in command data or events if not being modified. Property value constraints: `minItems: 1` default: null
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. May be null in command data or events if no configuration is to be or has been changed. default: null

Properties
unit/unit1/cash/configuration/currency ISO 4217 currency identifier [Ref. cashmanagement-1]. May only be modified in an exchange state if applicable. May be null if the unit is configured to store mixed currencies or non-cash items. Property value constraints: `pattern: ^[A-Z]{3}$` default: null
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. May be null in command data or events if not being modified. Property value constraints: `minimum: 0` default: null
unit/unit1/cash/configuration/highThreshold If specified, replenishmentStatus is 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 null, high is based on hardware sensors if supported - see hardwareSensors. May be null in command data or events if not being modified. Property value constraints: `minimum: 1` default: null
unit/unit1/cash/configuration/lowThreshold If specified, replenishmentStatus is 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 null, low is based on hardware sensors if supported - see hardwareSensors. May be null in command data or events if not being modified. Property value constraints: `minimum: 1` default: null
unit/unit1/cash/configuration/appLockIn If true, items cannot be accepted into the storage unit in Cash In operations. May be null in command data or events if not being modified. default: null
unit/unit1/cash/configuration/appLockOut If true, items cannot be dispensed from the storage unit in Cash Out operations. May be null in command data or events if not being modified. default: null
unit/unit1/cash/configuration/name Application configured name of the unit. May be null in command data or events if not being modified. default: null
unit/unit1/cash/configuration/maxRetracts If specified, this is the number of retract operations allowed into the unit. Only applies to retract units. If retractOperations equals this number, then no further retracts are allowed into this storage unit. If null in output, the maximum number is not limited by counts. May be null in command data or events if not being modified. Property value constraints: `minimum: 1` default: null

Properties
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. May be null in events if not changing. default: null
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. default: null
unit/unit1/cash/status/initial/unrecognized Count of unrecognized items handled by the cash interface. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
unit/unit1/cash/status/initial/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification. default: null
unit/unit1/cash/status/initial/type20USD1/fit Count of genuine cash items which are fit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
unit/unit1/cash/status/initial/type20USD1/unfit Count of genuine cash items which are unfit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
unit/unit1/cash/status/initial/type20USD1/suspect Count of suspected counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
unit/unit1/cash/status/initial/type20USD1/counterfeit Count of counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null

Properties
unit/unit1/cash/status/initial/type20USD1/inked Count of cash items which have been identified as ink stained. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
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. May be null if items have not or can not be moved from the storage unit by cash commands. default: null
unit/unit1/cash/status/out/presented The items dispensed from this storage unit which are or were customer accessible. Will be null if no items were presented. default: null
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. Will be null if no items were rejected. default: null
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. Will be null if no items were distributed. default: null
unit/unit1/cash/status/out/unknown The items dispensed from this storage unit which moved to an unknown position. Will be null if no items were unknown. default: null
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. Will be null if no items were stacked. default: null
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. Will be null if no items were diverted. default: null
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. Will be null if no items apply. default: null

Properties
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. May be null if items have not or can not be moved into the storage unit by cash commands. default: null
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. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
unit/unit1/cash/status/in/deposited The items deposited in the storage unit during a Cash In transaction. Can be null, if all values are 0. default: null
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*. Can be null, if all values are 0. default: null
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. Can be null, if all values are 0. default: null
unit/unit1/cash/status/in/distributed The items deposited in this storage unit originating from another storage unit but not rejected. Can be null, if all values are 0. default: null
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. Can be null, if all values are 0. default: null

Properties
unit/unit1/cash/status/accuracy Describes the accuracy of the counts reported by *out* and *in*. If null in Storage.GetStorage, the hardware is not capable of determining the accuracy, otherwise the following values are possible: • `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. default: null
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. May be null in events if not changing, otherwise 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. default: null
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 null in Storage.GetStorage, *status* and *replenishmentStatus* apply to both cash out and cash in operations. default: null
unit/unit1/card The card related contents, status and configuration of the unit. May be null if not applicable. default: null
unit/unit1/card/capabilities Indicates the card storage unit capabilities. This property can be null if a change is being reported using Storage.StorageChangedEvent or Storage.StorageThresholdEvent.

Properties
unit/unit1/card/capabilities/type The type of card storage. This property may be null in events if the type did not change, otherwise will be one of the following values: • `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. default: null
unit/unit1/card/capabilities/hardwareSensors Indicates whether the storage unit has hardware sensors that can detect threshold states. This property may be null in events if it did not change. default: null
unit/unit1/card/configuration Indicates the card storage unit configuration. This property can be null if the storage unit is being set using Storage.SetStorage, or a change is being reported using Storage.StorageChangedEvent or Storage.StorageThresholdEvent.
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 storage units and may be null in events if it did not change. default: null
unit/unit1/card/configuration/threshold If the threshold value is non zero, hardware sensors in the storage unit do not trigger Storage.StorageThresholdEvent events. This property may be null in events if it did not change. 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` default: null
unit/unit1/card/status Indicates the card storage unit status. This property can be null if a change is being reported using Storage.StorageChangedEvent or Storage.StorageThresholdEvent.
unit/unit1/card/status/initialCount The initial number of cards in the storage unit. This is only applicable to dispense type storage units. This property may be null in events if it did not change. This value is persistent. Property value constraints: `minimum: 0` default: null

Properties
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 and may be null in events if it did not change. Property value constraints: `minimum: 0` default: null
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 and may be null in events if it did not change. Property value constraints: `minimum: 0` default: null
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 null. The property may also be null in events if it did not change. 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. default: null
unit/unit1/check The check related contents, status and configuration of the unit. May be null if not applicable. default: null
unit/unit1/check/capabilities Indicates what the storage unit is capable of - this includes information which is a combination of that reported in WFS_INF_IPM_MEDIA_BIN_INFO and WFS_INF_IPM_MEDIA_BIN_CAPABILITIES in XFS 3.x. May be null in events if not changed. default: null
unit/unit1/check/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. May be null in command data and events if not changing. default: null

Properties
unit/unit1/check/capabilities/types/mediaIn The unit can accept items during Media In transactions. May be null in command data and events if not changing. default: null
unit/unit1/check/capabilities/types/retract Retract unit. Items can be retracted into this unit using Check.RetractMedia. May be null in command data and events if not changing. default: null
unit/unit1/check/capabilities/sensors The types of sensor the unit has. May be null in command data and events if not changing. default: null
unit/unit1/check/capabilities/sensors/empty The unit contains a hardware sensor which reports when the unit is empty. May be null in command data and events if not changing. default: null
unit/unit1/check/capabilities/sensors/high The unit contains a hardware sensor which reports when the unit is nearly full. May be null in command data and events if not changing. default: null
unit/unit1/check/capabilities/sensors/full The unit contains a hardware sensor which reports when the unit is full. May be null in command data and events if not changing. default: null
unit/unit1/check/configuration Indicates what the storage unit is configured to do - where applicable the supported options can be derived from capabilities. May be null in command data and events if not being modified.
unit/unit1/check/configuration/binID An application defined Storage Unit Identifier. This may be null in events if not changing. default: null
unit/unit1/check/configuration/highThreshold If specified, replenishmentStatus is set to *high* if the total number of items in the storage unit is greater than this number. May be null in command data and events if not being modified. Property value constraints: `minimum: 1` default: null
unit/unit1/check/configuration/retractHighThreshold If specified and the storage unit is configured as *retract*, replenishmentStatus is set to *high* if the total number of retract operations in the storage unit is greater than this number. May be null in command data and events if not being modified. Property value constraints: `minimum: 0` default: null
unit/unit1/check/status Indicates the storage unit status. May be null in events where status has not changed. default: null

Properties
unit/unit1/check/status/index Assigned by the Service. Will be a unique number which can be used to determine *usBinNumber* 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/check/status/initial The check related counts as set at the last replenishment. May be null in events where status has not changed. default: null
unit/unit1/check/status/initial/mediaInCount Count of items added to the storage unit due to Check operations. If the number of items is not counted this is not reported and *retractOperations* is incremented as items are added to the unit. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
unit/unit1/check/status/initial/count Total number of items added to the storage unit due to any operations. If the number of items is not counted this is not reported and *retractOperations* is incremented as items are added to the unit. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
unit/unit1/check/status/initial/retractOperations Total number of operations which resulted in items being retracted to the storage unit. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
unit/unit1/check/status/in The check items added to the unit since the last replenishment. May be null in events where status has not changed. default: null
unit/unit1/check/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. May be null in command data and events if not changing, otherwise 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 a full sensor or counts. • `high` - The storage unit is almost full (either high sensor based or exceeded the highThreshold or retractHighThreshold). • `empty` - The storage unit is empty, or insufficient items in the storage unit are preventing further dispense operations. If the storage unit has the empty sensor, this state is not set by counts. default: null

22.4 Unsolicited Messages

22.4.1 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 2.0)	Type	Required
{		
"*unit1*": {	object	
"id": "RC1",	string, null	
"positionName": "Top Right",	string, null	
"capacity": 100,	integer, null	
"status": "ok",	string, null	
"serialNumber": "ABCD1234",	string, null	
"cash": {	object, null	
"capabilities": {	object, null	
"types": {	object, null	
"cashIn": true,	boolean, null	
"cashOut": false,	boolean, null	
"replenishment": false,	boolean, null	
"cashInRetract": false,	boolean, null	
"cashOutRetract": false,	boolean, null	
"reject": false	boolean, null	
},		
"items": {	object, null	
"fit": false,	boolean, null	
"unfit": false,	boolean, null	
"unrecognized": false,	boolean, null	
"counterfeit": false,	boolean, null	
"suspect": false,	boolean, null	
"inked": false,	boolean, null	
"coupon": false,	boolean, null	
"document": false	boolean, null	
},		
"hardwareSensors": false,	boolean, null	
"retractAreas": 1,	integer, null	
"retractThresholds": false,	boolean, null	

Payload (version 2.0)	Type	Required
"cashItems": ["type20USD1", "type50USD1"]	array (string), null	
},		
"configuration": {	object, null	
"types": See unit1/cash/capabilities/types properties	object, null	
"items": See unit1/cash/capabilities/items properties	object, null	
"currency": "USD",	string, null	
"value": 20.00,	number, null	
"highThreshold": 500,	integer, null	
"lowThreshold": 10,	integer, null	
"appLockIn": false,	boolean, null	
"appLockOut": false,	boolean, null	
"cashItems": See unit1/cash/capabilities/cashItems,	array (string), null	
"name": "$10",	string, null	
"maxRetracts": 5	integer, null	
},		
"status": {	object, null	
"index": 4,	integer	✓
"initial": {	object, null	
"unrecognized": 5,	integer, null	
"*type20USD1*": {	object, null	
"fit": 15,	integer, null	
"unfit": 0,	integer, null	
"suspect": 0,	integer, null	
"counterfeit": 0,	integer, null	
"inked": 0	integer, null	
},		
"*type50USD1*": See *unit1/cash/status/initial/type20USD1* properties	object, null	
},		
"out": {	object, null	
"presented": See unit1/cash/status/initial properties	object, null	
"rejected": See unit1/cash/status/initial properties	object, null	
"distributed": See unit1/cash/status/initial properties	object, null	
"unknown": See unit1/cash/status/initial properties	object, null	
"stacked": See unit1/cash/status/initial properties	object, null	
"diverted": See unit1/cash/status/initial properties	object, null	
"transport": See unit1/cash/status/initial properties	object, null	

Payload (version 2.0)	Type	Required
},		
"in": {	object, null	
"retractOperations": 15,	integer, null	
"deposited": See unit1/cash/status/initial properties	object, null	
"retracted": See unit1/cash/status/initial properties	object, null	
"rejected": See unit1/cash/status/initial properties	object, null	
"distributed": See unit1/cash/status/initial properties	object, null	
"transport": See unit1/cash/status/initial properties	object, null	
},		
"accuracy": "accurate",	string, null	
"replenishmentStatus": "ok",	string, null	
"operationStatus": "dispenseInoperative"	string, null	
}		
},		
"card": {	object, null	
"capabilities": {	object, null	
"type": "retain",	string, null	
"hardwareSensors": true	boolean, null	
},		
"configuration": {	object, null	
"cardID": "LoyaltyCard",	string, null	
"threshold": 10	integer, null	
},		
"status": {	object, null	
"initialCount": 0,	integer, null	
"count": 0,	integer, null	
"retainCount": 0,	integer, null	
"replenishmentStatus": "ok"	string, null	
}		
},		
"check": {	object, null	
"capabilities": {	object, null	
"types": {	object, null	
"mediaIn": true,	boolean, null	
"retract": false	boolean, null	
},		
"sensors": {	object, null	
"empty": false,	boolean, null	

Payload (version 2.0)	Type	Required
"high": false,	boolean, null	
"full": false	boolean, null	
}		
},		
"configuration": {	object, null	
"types": See unit1/check/capabilities/types properties	object, null	
"binID": "My check bin",	string, null	
"highThreshold": 500,	integer, null	
"retractHighThreshold": 5	integer, null	
},		
"status": {	object, null	
"index": 4,	integer	✓
"initial": {	object, null	
"mediaInCount": 100,	integer, null	
"count": 150,	integer, null	
"retractOperations": 15	integer, null	
},		
"in": See unit1/check/status/initial properties	object, null	
"replenishmentStatus": "high"	string, null	
}		
}		
}		
}		

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. May be null if not applicable.

Property value constraints:

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

default: null

unit1/positionName

Fixed physical name for the position. May be null if not applicable.

default: null

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, null means capacity is not applicable.

Property value constraints:

```
minimum: 0
```

default: null

Properties
unit1/status The state of the unit. This property may be null in events if the state did not change, otherwise 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. default: null
unit1/serialNumber The storage unit's serial number if it can be read electronically. May be null if not applicable. default: null
unit1/cash The cash related contents, status and configuration of the unit. May be null if not applicable. default: null
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. This may be null in events if capabilities have not changed. default: null
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. May be null in command data or events if not changed or being changed. default: null
unit1/cash/capabilities/types/cashIn The unit can accept cash items. If *cashOut* is also true then the unit can recycle. May be null in command data or events if not changed or being changed. default: null
unit1/cash/capabilities/types/cashOut The unit can dispense cash items. If *cashIn* is also true then the unit can recycle. May be null in command data or events if not changed or being changed. default: null
unit1/cash/capabilities/types/replenishment Replenishment container. A storage unit can be refilled from or emptied to a replenishment container. May be null in command data or events if not changed or being changed. default: null
unit1/cash/capabilities/types/cashInRetract Retract unit. Items can be retracted into this unit during Cash In operations. May be null in command data or events if not changed or being changed. default: null
unit1/cash/capabilities/types/cashOutRetract Retract unit. Items can be retracted into this unit during Cash Out operations. May be null in command data or events if not changed or being changed. default: null

Properties
unit1/cash/capabilities/types/reject Reject unit. Items can be rejected into this unit. May be null in command data or events if not changed or being changed. default: null
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. May be null in command data if not being changed. May be null in command data or events if not changed or being changed. default: null
unit1/cash/capabilities/items/fit The storage unit can store cash items which are fit for recycling. May be null in command data or events if not changed or being changed. default: null
unit1/cash/capabilities/items/unfit The storage unit can store cash items which are unfit for recycling. May be null in command data or events if not changed or being changed. default: false
unit1/cash/capabilities/items/unrecognized The storage unit can store unrecognized cash items. May be null in command data or events if not changed or being changed. default: null
unit1/cash/capabilities/items/counterfeit The storage unit can store counterfeit cash items. May be null in command data or events if not changed or being changed. default: null
unit1/cash/capabilities/items/suspect The storage unit can store suspect counterfeit cash items. May be null in command data or events if not changed or being changed. default: null
unit1/cash/capabilities/items/inked The storage unit can store cash items which have been identified as ink stained. May be null in command data or events if not changed or being changed. default: null
unit1/cash/capabilities/items/coupon Storage unit containing coupons or advertising material. May be null in command data or events if not changed or being changed. default: null
unit1/cash/capabilities/items/document Storage unit containing documents. May be null in command data or events if not changed or being changed. default: null
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. May be null in command data or events if not changed or being changed. default: null

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. May be null if items can not be retracted into this storage unit or in events if not changed or being changed. Property value constraints: `minimum: 1` default: null
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. May be null if items can not be retracted into this storage unit or in events if not changed or being changed. default: null
unit1/cash/capabilities/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. May be null in command data or events if not being modified. Property value constraints: `minItems: 1` default: null
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. May be null in command data or events if no configuration is to be or has been changed. default: null
unit1/cash/configuration/currency ISO 4217 currency identifier [Ref. cashmanagement-1]. May only be modified in an exchange state if applicable. May be null if the unit is configured to store mixed currencies or non-cash items. Property value constraints: `pattern: ^[A-Z]{3}$` default: null
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. May be null in command data or events if not being modified. Property value constraints: `minimum: 0` default: null
unit1/cash/configuration/highThreshold If specified, replenishmentStatus is 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 null, high is based on hardware sensors if supported - see hardwareSensors. May be null in command data or events if not being modified. Property value constraints: `minimum: 1` default: null

Properties
unit1/cash/configuration/lowThreshold If specified, replenishmentStatus is 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 null, low is based on hardware sensors if supported - see hardwareSensors. May be null in command data or events if not being modified. Property value constraints: `minimum: 1` default: null
unit1/cash/configuration/appLockIn If true, items cannot be accepted into the storage unit in Cash In operations. May be null in command data or events if not being modified. default: null
unit1/cash/configuration/appLockOut If true, items cannot be dispensed from the storage unit in Cash Out operations. May be null in command data or events if not being modified. default: null
unit1/cash/configuration/name Application configured name of the unit. May be null in command data or events if not being modified. default: null
unit1/cash/configuration/maxRetracts If specified, this is the number of retract operations allowed into the unit. Only applies to retract units. If retractOperations equals this number, then no further retracts are allowed into this storage unit. If null in output, the maximum number is not limited by counts. May be null in command data or events if not being modified. Property value constraints: `minimum: 1` default: null
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. May be null in events if not changing. default: null
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. default: null
unit1/cash/status/initial/unrecognized Count of unrecognized items handled by the cash interface. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null

Properties
unit1/cash/status/initial/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification. default: null
unit1/cash/status/initial/type20USD1/fit Count of genuine cash items which are fit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
unit1/cash/status/initial/type20USD1/unfit Count of genuine cash items which are unfit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
unit1/cash/status/initial/type20USD1/suspect Count of suspected counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
unit1/cash/status/initial/type20USD1/counterfeit Count of counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
unit1/cash/status/initial/type20USD1/inked Count of cash items which have been identified as ink stained. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
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. May be null if items have not or can not be moved from the storage unit by cash commands. default: null
unit1/cash/status/out/presented The items dispensed from this storage unit which are or were customer accessible. Will be null if no items were presented. default: null

Properties
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. Will be null if no items were rejected. default: null
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. Will be null if no items were distributed. default: null
unit1/cash/status/out/unknown The items dispensed from this storage unit which moved to an unknown position. Will be null if no items were unknown. default: null
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. Will be null if no items were stacked. default: null
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. Will be null if no items were diverted. default: null
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. Will be null if no items apply. default: null
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. May be null if items have not or can not be moved into the storage unit by cash commands. default: null
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. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
unit1/cash/status/in/deposited The items deposited in the storage unit during a Cash In transaction. Can be null, if all values are 0. default: null

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*. Can be null, if all values are 0. default: null
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. Can be null, if all values are 0. default: null
unit1/cash/status/in/distributed The items deposited in this storage unit originating from another storage unit but not rejected. Can be null, if all values are 0. default: null
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. Can be null, if all values are 0. default: null
unit1/cash/status/accuracy Describes the accuracy of the counts reported by *out* and *in*. If null in Storage.GetStorage, the hardware is not capable of determining the accuracy, otherwise the following values are possible: • `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. default: null
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. May be null in events if not changing, otherwise 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. default: null

Properties
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 null in Storage.GetStorage, *status* and *replenishmentStatus* apply to both cash out and cash in operations. default: null
unit1/card The card related contents, status and configuration of the unit. May be null if not applicable. default: null
unit1/card/capabilities Indicates the card storage unit capabilities. This property can be null if a change is being reported using Storage.StorageChangedEvent or Storage.StorageThresholdEvent.
unit1/card/capabilities/type The type of card storage. This property may be null in events if the type did not change, otherwise will be one of the following values: • `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. default: null
unit1/card/capabilities/hardwareSensors Indicates whether the storage unit has hardware sensors that can detect threshold states. This property may be null in events if it did not change. default: null
unit1/card/configuration Indicates the card storage unit configuration. This property can be null if the storage unit is being set using Storage.SetStorage, or a change is being reported using Storage.StorageChangedEvent or Storage.StorageThresholdEvent.
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 storage units and may be null in events if it did not change. default: null
unit1/card/configuration/threshold If the threshold value is non zero, hardware sensors in the storage unit do not trigger Storage.StorageThresholdEvent events. This property may be null in events if it did not change. 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` default: null
unit1/card/status Indicates the card storage unit status. This property can be null if a change is being reported using Storage.StorageChangedEvent or Storage.StorageThresholdEvent.

Properties
unit1/card/status/initialCount The initial number of cards in the storage unit. This is only applicable to dispense type storage units. This property may be null in events if it did not change. This value is persistent. Property value constraints: `minimum: 0` default: null
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 and may be null in events if it did not change. Property value constraints: `minimum: 0` default: null
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 and may be null in events if it did not change. Property value constraints: `minimum: 0` default: null
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 null. The property may also be null in events if it did not change. 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. default: null
unit1/check The check related contents, status and configuration of the unit. May be null if not applicable. default: null

Properties
unit1/check/capabilities Indicates what the storage unit is capable of - this includes information which is a combination of that reported in WFS_INF_IPM_MEDIA_BIN_INFO and WFS_INF_IPM_MEDIA_BIN_CAPABILITIES in XFS 3.x. May be null in events if not changed. default: null
unit1/check/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. May be null in command data and events if not changing. default: null
unit1/check/capabilities/types/mediaIn The unit can accept items during Media In transactions. May be null in command data and events if not changing. default: null
unit1/check/capabilities/types/retract Retract unit. Items can be retracted into this unit using Check.RetractMedia. May be null in command data and events if not changing. default: null
unit1/check/capabilities/sensors The types of sensor the unit has. May be null in command data and events if not changing. default: null
unit1/check/capabilities/sensors/empty The unit contains a hardware sensor which reports when the unit is empty. May be null in command data and events if not changing. default: null
unit1/check/capabilities/sensors/high The unit contains a hardware sensor which reports when the unit is nearly full. May be null in command data and events if not changing. default: null
unit1/check/capabilities/sensors/full The unit contains a hardware sensor which reports when the unit is full. May be null in command data and events if not changing. default: null
unit1/check/configuration Indicates what the storage unit is configured to do - where applicable the supported options can be derived from capabilities. May be null in command data and events if not being modified.
unit1/check/configuration/binID An application defined Storage Unit Identifier. This may be null in events if not changing. default: null
unit1/check/configuration/highThreshold If specified, replenishmentStatus is set to *high* if the total number of items in the storage unit is greater than this number. May be null in command data and events if not being modified. Property value constraints: `minimum: 1` default: null

Properties
unit1/check/configuration/retractHighThreshold If specified and the storage unit is configured as *retract*, replenishmentStatus is set to *high* if the total number of retract operations in the storage unit is greater than this number. May be null in command data and events if not being modified. Property value constraints: `minimum: 0` default: null
unit1/check/status Indicates the storage unit status. May be null in events where status has not changed. default: null
unit1/check/status/index Assigned by the Service. Will be a unique number which can be used to determine *usBinNumber* 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/check/status/initial The check related counts as set at the last replenishment. May be null in events where status has not changed. default: null
unit1/check/status/initial/mediaInCount Count of items added to the storage unit due to Check operations. If the number of items is not counted this is not reported and *retractOperations* is incremented as items are added to the unit. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
unit1/check/status/initial/count Total number of items added to the storage unit due to any operations. If the number of items is not counted this is not reported and *retractOperations* is incremented as items are added to the unit. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
unit1/check/status/initial/retractOperations Total number of operations which resulted in items being retracted to the storage unit. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
unit1/check/status/in The check items added to the unit since the last replenishment. May be null in events where status has not changed. default: null

Properties
unit1/check/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. May be null in command data and events if not changing, otherwise 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 a full sensor or counts. • `high` - The storage unit is almost full (either high sensor based or exceeded the highThreshold or retractHighThreshold). • `empty` - The storage unit is empty, or insufficient items in the storage unit are preventing further dispense operations. If the storage unit has the empty sensor, this state is not set by counts. default: null

22.4.2 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 2.0)	Type	Required
{		
"unit1": {	object	
"id": "RC1",	string, null	
"positionName": "Top Right",	string, null	
"capacity": 100,	integer, null	
"status": "ok",	string, null	
"serialNumber": "ABCD1234",	string, null	
"cash": {	object, null	
"capabilities": {	object, null	
"types": {	object, null	
"cashIn": true,	boolean, null	
"cashOut": false,	boolean, null	
"replenishment": false,	boolean, null	
"cashInRetract": false,	boolean, null	
"cashOutRetract": false,	boolean, null	
"reject": false	boolean, null	
},		
"items": {	object, null	
"fit": false,	boolean, null	
"unfit": false,	boolean, null	
"unrecognized": false,	boolean, null	
"counterfeit": false,	boolean, null	
"suspect": false,	boolean, null	
"inked": false,	boolean, null	
"coupon": false,	boolean, null	
"document": false	boolean, null	
},		
"hardwareSensors": false,	boolean, null	
"retractAreas": 1,	integer, null	
"retractThresholds": false,	boolean, null	
"cashItems": ["type20USD1", "type50USD1"]	array (string), null	
},		
"configuration": {	object, null	
"types": See unit1/cash/capabilities/types properties	object, null	

Payload (version 2.0)	Type	Required
"items": See unit1/cash/capabilities/items properties	object, null	
"currency": "USD",	string, null	
"value": 20.00,	number, null	
"highThreshold": 500,	integer, null	
"lowThreshold": 10,	integer, null	
"appLockIn": false,	boolean, null	
"appLockOut": false,	boolean, null	
"cashItems": See unit1/cash/capabilities/cashItems,	array (string), null	
"name": "$10",	string, null	
"maxRetracts": 5	integer, null	
},		
"status": {	object, null	
"index": 4,	integer	✓
"initial": {	object, null	
"unrecognized": 5,	integer, null	
"*type20USD1*": {	object, null	
"fit": 15,	integer, null	
"unfit": 0,	integer, null	
"suspect": 0,	integer, null	
"counterfeit": 0,	integer, null	
"inked": 0	integer, null	
},		
"*type50USD1*": See *unit1/cash/status/initial/type20USD1* properties	object, null	
},		
"out": {	object, null	
"presented": See unit1/cash/status/initial properties	object, null	
"rejected": See unit1/cash/status/initial properties	object, null	
"distributed": See unit1/cash/status/initial properties	object, null	
"unknown": See unit1/cash/status/initial properties	object, null	
"stacked": See unit1/cash/status/initial properties	object, null	
"diverted": See unit1/cash/status/initial properties	object, null	
"transport": See unit1/cash/status/initial properties	object, null	
},		
"in": {	object, null	
"retractOperations": 15,	integer, null	
"deposited": See unit1/cash/status/initial properties	object, null	

Payload (version 2.0)	Type	Required
"retracted": See unit1/cash/status/initial properties	object, null	
"rejected": See unit1/cash/status/initial properties	object, null	
"distributed": See unit1/cash/status/initial properties	object, null	
"transport": See unit1/cash/status/initial properties	object, null	
},		
"accuracy": "accurate",	string, null	
"replenishmentStatus": "ok",	string, null	
"operationStatus": "dispenseInoperative"	string, null	
}		
},		
"card": {	object, null	
"capabilities": {	object, null	
"type": "retain",	string, null	
"hardwareSensors": true	boolean, null	
},		
"configuration": {	object, null	
"cardID": "LoyaltyCard",	string, null	
"threshold": 10	integer, null	
},		
"status": {	object, null	
"initialCount": 0,	integer, null	
"count": 0,	integer, null	
"retainCount": 0,	integer, null	
"replenishmentStatus": "ok"	string, null	
}		
},		
"check": {	object, null	
"capabilities": {	object, null	
"types": {	object, null	
"mediaIn": true,	boolean, null	
"retract": false	boolean, null	
},		
"sensors": {	object, null	
"empty": false,	boolean, null	
"high": false,	boolean, null	
"full": false	boolean, null	
}		
},		

Payload (version 2.0)	Type	Required
"configuration": {	object, null	
"types": See unit1/check/capabilities/types properties	object, null	
"binID": "My check bin",	string, null	
"highThreshold": 500,	integer, null	
"retractHighThreshold": 5	integer, null	
},		
"status": {	object, null	
"index": 4,	integer	✓
"initial": {	object, null	
"mediaInCount": 100,	integer, null	
"count": 150,	integer, null	
"retractOperations": 15	integer, null	
},		
"in": See unit1/check/status/initial properties	object, null	
"replenishmentStatus": "high"	string, null	
}		
}		
}		
}		

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. May be null if not applicable.

Property value constraints:

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

default: null

unit1/positionName

Fixed physical name for the position. May be null if not applicable.

default: null

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, null means capacity is not applicable.

Property value constraints:

```
minimum: 0
```

default: null

Properties
unit1/status The state of the unit. This property may be null in events if the state did not change, otherwise 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. default: null
unit1/serialNumber The storage unit's serial number if it can be read electronically. May be null if not applicable. default: null
unit1/cash The cash related contents, status and configuration of the unit. May be null if not applicable. default: null
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. This may be null in events if capabilities have not changed. default: null
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. May be null in command data or events if not changed or being changed. default: null
unit1/cash/capabilities/types/cashIn The unit can accept cash items. If *cashOut* is also true then the unit can recycle. May be null in command data or events if not changed or being changed. default: null
unit1/cash/capabilities/types/cashOut The unit can dispense cash items. If *cashIn* is also true then the unit can recycle. May be null in command data or events if not changed or being changed. default: null
unit1/cash/capabilities/types/replenishment Replenishment container. A storage unit can be refilled from or emptied to a replenishment container. May be null in command data or events if not changed or being changed. default: null
unit1/cash/capabilities/types/cashInRetract Retract unit. Items can be retracted into this unit during Cash In operations. May be null in command data or events if not changed or being changed. default: null
unit1/cash/capabilities/types/cashOutRetract Retract unit. Items can be retracted into this unit during Cash Out operations. May be null in command data or events if not changed or being changed. default: null

Properties
unit1/cash/capabilities/types/reject Reject unit. Items can be rejected into this unit. May be null in command data or events if not changed or being changed. default: null
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. May be null in command data if not being changed. May be null in command data or events if not changed or being changed. default: null
unit1/cash/capabilities/items/fit The storage unit can store cash items which are fit for recycling. May be null in command data or events if not changed or being changed. default: null
unit1/cash/capabilities/items/unfit The storage unit can store cash items which are unfit for recycling. May be null in command data or events if not changed or being changed. default: false
unit1/cash/capabilities/items/unrecognized The storage unit can store unrecognized cash items. May be null in command data or events if not changed or being changed. default: null
unit1/cash/capabilities/items/counterfeit The storage unit can store counterfeit cash items. May be null in command data or events if not changed or being changed. default: null
unit1/cash/capabilities/items/suspect The storage unit can store suspect counterfeit cash items. May be null in command data or events if not changed or being changed. default: null
unit1/cash/capabilities/items/inked The storage unit can store cash items which have been identified as ink stained. May be null in command data or events if not changed or being changed. default: null
unit1/cash/capabilities/items/coupon Storage unit containing coupons or advertising material. May be null in command data or events if not changed or being changed. default: null
unit1/cash/capabilities/items/document Storage unit containing documents. May be null in command data or events if not changed or being changed. default: null
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. May be null in command data or events if not changed or being changed. default: null

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. May be null if items can not be retracted into this storage unit or in events if not changed or being changed. Property value constraints: `minimum: 1` default: null
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. May be null if items can not be retracted into this storage unit or in events if not changed or being changed. default: null
unit1/cash/capabilities/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. May be null in command data or events if not being modified. Property value constraints: `minItems: 1` default: null
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. May be null in command data or events if no configuration is to be or has been changed. default: null
unit1/cash/configuration/currency ISO 4217 currency identifier [Ref. cashmanagement-1]. May only be modified in an exchange state if applicable. May be null if the unit is configured to store mixed currencies or non-cash items. Property value constraints: `pattern: ^[A-Z]{3}$` default: null
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. May be null in command data or events if not being modified. Property value constraints: `minimum: 0` default: null
unit1/cash/configuration/highThreshold If specified, replenishmentStatus is 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 null, high is based on hardware sensors if supported - see hardwareSensors. May be null in command data or events if not being modified. Property value constraints: `minimum: 1` default: null

Properties
unit1/cash/configuration/lowThreshold If specified, replenishmentStatus is 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 null, low is based on hardware sensors if supported - see hardwareSensors. May be null in command data or events if not being modified. Property value constraints: `minimum: 1` default: null
unit1/cash/configuration/appLockIn If true, items cannot be accepted into the storage unit in Cash In operations. May be null in command data or events if not being modified. default: null
unit1/cash/configuration/appLockOut If true, items cannot be dispensed from the storage unit in Cash Out operations. May be null in command data or events if not being modified. default: null
unit1/cash/configuration/name Application configured name of the unit. May be null in command data or events if not being modified. default: null
unit1/cash/configuration/maxRetracts If specified, this is the number of retract operations allowed into the unit. Only applies to retract units. If retractOperations equals this number, then no further retracts are allowed into this storage unit. If null in output, the maximum number is not limited by counts. May be null in command data or events if not being modified. Property value constraints: `minimum: 1` default: null
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. May be null in events if not changing. default: null
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. default: null
unit1/cash/status/initial/unrecognized Count of unrecognized items handled by the cash interface. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null

Properties
unit1/cash/status/initial/type20USD1 (example name) Counts of a given cash item (as reported by CashManagement.GetBankNoteTypes) broken down by classification. default: null
unit1/cash/status/initial/type20USD1/fit Count of genuine cash items which are fit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
unit1/cash/status/initial/type20USD1/unfit Count of genuine cash items which are unfit for recycling. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
unit1/cash/status/initial/type20USD1/suspect Count of suspected counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
unit1/cash/status/initial/type20USD1/counterfeit Count of counterfeit cash items. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
unit1/cash/status/initial/type20USD1/inked Count of cash items which have been identified as ink stained. May be null in command data and events if not changed or not to be changed. Property value constraints: `minimum: 0` default: null
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. May be null if items have not or can not be moved from the storage unit by cash commands. default: null
unit1/cash/status/out/presented The items dispensed from this storage unit which are or were customer accessible. Will be null if no items were presented. default: null

Properties
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. Will be null if no items were rejected. default: null
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. Will be null if no items were distributed. default: null
unit1/cash/status/out/unknown The items dispensed from this storage unit which moved to an unknown position. Will be null if no items were unknown. default: null
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. Will be null if no items were stacked. default: null
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. Will be null if no items were diverted. default: null
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. Will be null if no items apply. default: null
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. May be null if items have not or can not be moved into the storage unit by cash commands. default: null
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. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
unit1/cash/status/in/deposited The items deposited in the storage unit during a Cash In transaction. Can be null, if all values are 0. default: null

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*. Can be null, if all values are 0. default: null
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. Can be null, if all values are 0. default: null
unit1/cash/status/in/distributed The items deposited in this storage unit originating from another storage unit but not rejected. Can be null, if all values are 0. default: null
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. Can be null, if all values are 0. default: null
unit1/cash/status/accuracy Describes the accuracy of the counts reported by *out* and *in*. If null in Storage.GetStorage, the hardware is not capable of determining the accuracy, otherwise the following values are possible: • `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. default: null
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. May be null in events if not changing, otherwise 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. default: null

Properties
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 null in Storage.GetStorage, *status* and *replenishmentStatus* apply to both cash out and cash in operations. default: null
unit1/card The card related contents, status and configuration of the unit. May be null if not applicable. default: null
unit1/card/capabilities Indicates the card storage unit capabilities. This property can be null if a change is being reported using Storage.StorageChangedEvent or Storage.StorageThresholdEvent.
unit1/card/capabilities/type The type of card storage. This property may be null in events if the type did not change, otherwise will be one of the following values: • `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. default: null
unit1/card/capabilities/hardwareSensors Indicates whether the storage unit has hardware sensors that can detect threshold states. This property may be null in events if it did not change. default: null
unit1/card/configuration Indicates the card storage unit configuration. This property can be null if the storage unit is being set using Storage.SetStorage, or a change is being reported using Storage.StorageChangedEvent or Storage.StorageThresholdEvent.
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 storage units and may be null in events if it did not change. default: null
unit1/card/configuration/threshold If the threshold value is non zero, hardware sensors in the storage unit do not trigger Storage.StorageThresholdEvent events. This property may be null in events if it did not change. 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` default: null
unit1/card/status Indicates the card storage unit status. This property can be null if a change is being reported using Storage.StorageChangedEvent or Storage.StorageThresholdEvent.

Properties
unit1/card/status/initialCount The initial number of cards in the storage unit. This is only applicable to dispense type storage units. This property may be null in events if it did not change. This value is persistent. Property value constraints: `minimum: 0` default: null
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 and may be null in events if it did not change. Property value constraints: `minimum: 0` default: null
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 and may be null in events if it did not change. Property value constraints: `minimum: 0` default: null
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 null. The property may also be null in events if it did not change. 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. default: null
unit1/check The check related contents, status and configuration of the unit. May be null if not applicable. default: null

Properties
unit1/check/capabilities Indicates what the storage unit is capable of - this includes information which is a combination of that reported in WFS_INF_IPM_MEDIA_BIN_INFO and WFS_INF_IPM_MEDIA_BIN_CAPABILITIES in XFS 3.x. May be null in events if not changed. default: null
unit1/check/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. May be null in command data and events if not changing. default: null
unit1/check/capabilities/types/mediaIn The unit can accept items during Media In transactions. May be null in command data and events if not changing. default: null
unit1/check/capabilities/types/retract Retract unit. Items can be retracted into this unit using Check.RetractMedia. May be null in command data and events if not changing. default: null
unit1/check/capabilities/sensors The types of sensor the unit has. May be null in command data and events if not changing. default: null
unit1/check/capabilities/sensors/empty The unit contains a hardware sensor which reports when the unit is empty. May be null in command data and events if not changing. default: null
unit1/check/capabilities/sensors/high The unit contains a hardware sensor which reports when the unit is nearly full. May be null in command data and events if not changing. default: null
unit1/check/capabilities/sensors/full The unit contains a hardware sensor which reports when the unit is full. May be null in command data and events if not changing. default: null
unit1/check/configuration Indicates what the storage unit is configured to do - where applicable the supported options can be derived from capabilities. May be null in command data and events if not being modified.
unit1/check/configuration/binID An application defined Storage Unit Identifier. This may be null in events if not changing. default: null
unit1/check/configuration/highThreshold If specified, replenishmentStatus is set to *high* if the total number of items in the storage unit is greater than this number. May be null in command data and events if not being modified. Property value constraints: `minimum: 1` default: null

Properties
unit1/check/configuration/retractHighThreshold If specified and the storage unit is configured as *retract*, replenishmentStatus is set to *high* if the total number of retract operations in the storage unit is greater than this number. May be null in command data and events if not being modified. Property value constraints: `minimum: 0` default: null
unit1/check/status Indicates the storage unit status. May be null in events where status has not changed. default: null
unit1/check/status/index Assigned by the Service. Will be a unique number which can be used to determine *usBinNumber* 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/check/status/initial The check related counts as set at the last replenishment. May be null in events where status has not changed. default: null
unit1/check/status/initial/mediaInCount Count of items added to the storage unit due to Check operations. If the number of items is not counted this is not reported and *retractOperations* is incremented as items are added to the unit. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
unit1/check/status/initial/count Total number of items added to the storage unit due to any operations. If the number of items is not counted this is not reported and *retractOperations* is incremented as items are added to the unit. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
unit1/check/status/initial/retractOperations Total number of operations which resulted in items being retracted to the storage unit. May be null in command data and events if not changing. Property value constraints: `minimum: 0` default: null
unit1/check/status/in The check items added to the unit since the last replenishment. May be null in events where status has not changed. default: null

Properties
unit1/check/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. May be null in command data and events if not changing, otherwise 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 a full sensor or counts. • `high` - The storage unit is almost full (either high sensor based or exceeded the highThreshold or retractHighThreshold). • `empty` - The storage unit is empty, or insufficient items in the storage unit are preventing further dispense operations. If the storage unit has the empty sensor, this state is not set by counts. default: null

23. 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.

23.1 General Information

23.1.1 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.

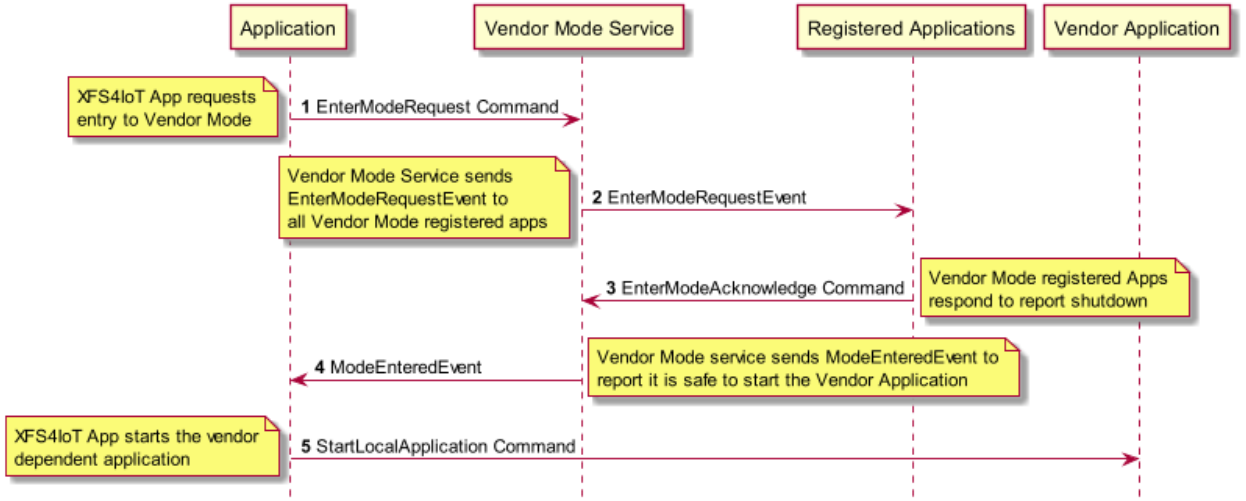
The Vendor Mode Service is used in conjunction with the Vendor Application service. The Vendor Mode Service is responsible for coordinating the release of resources from other services, while the Vendor Application service is responsible for starting the vendor dependent application. The VendorApplication.StartLocalApplication 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.

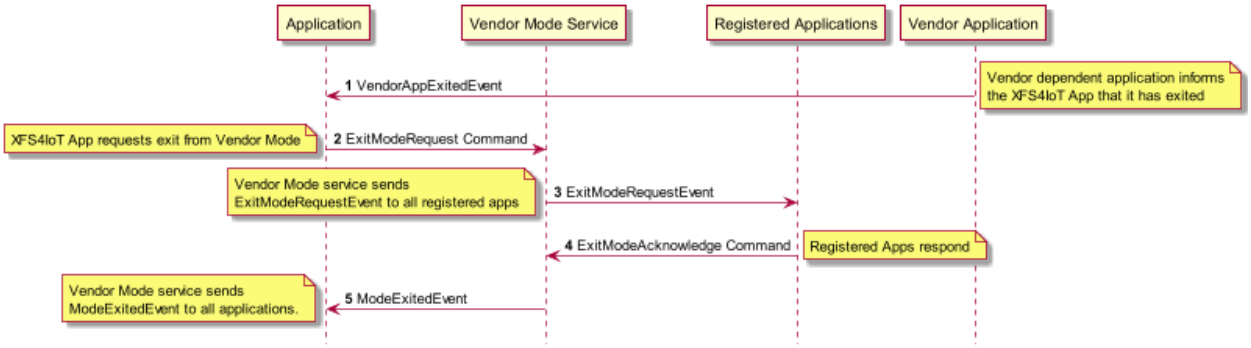
Vendor Mode Entry triggered by an XFS Application



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.

23.2 Command Messages

23.2.1 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 2.0)	Type	Required
{		
"appName": "ACME Monitoring app"	string	✓
}		

Properties

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 2.0)
This message does not define any properties.

Event Messages

None

23.2.2 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 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)
`This message does not define any properties.`

Event Messages

None

23.2.3 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 2.0)
This message does not define any properties.

Completion Message

Payload (version 2.0)
This message does not define any properties.

Event Messages

None

23.2.4 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 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)
`This message does not define any properties.`

Event Messages

None

23.2.5 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 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)
`This message does not define any properties.`

Event Messages

None

23.3 Unsolicited Messages

23.3.1 VendorMode.EnterModeRequestEvent

This service event is used to indicate the request to enter Vendor Mode.

Unsolicited Message

Payload (version 2.0)
This message does not define any properties.

23.3.2 VendorMode.ExitModeRequestEvent

This service event is used to indicate the request to exit Vendor Mode.

Unsolicited Message

Payload (version 2.0)
`This message does not define any properties.`

23.3.3 VendorMode.ModeEnteredEvent

This event is used to indicate that the system has entered Vendor Mode.

Unsolicited Message

Payload (version 2.0)
`This message does not define any properties.`

23.3.4 VendorMode.ModeExitedEvent

This event is used to indicate that the system has exited Vendor Mode.

Unsolicited Message

Payload (version 2.0)	Type	Required
{		
"connectedApplications": ["Application1", "Application2"]	array (string)	✓
}		

Properties

connectedApplications
List of applications that have not shut down.

24. 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.

24.1 General Information

24.1.1 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.

24.2 Command Messages

24.2.1 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 2.0)	Type	Required
{		
"appName": "ACME vendor app",	string	✓
"accessLevel": "basic"	string, null	
}		

Properties

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 (null) 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.

default: null

Completion Message

Payload (version 2.0)
This message does not define any properties.

Event Messages

None

24.2.2 VendorApplication.GetActiveInterface

This command is used to retrieve the interface that should be used by the vendor dependent application.

Command Message

Payload (version 2.0)
`This message does not define any properties.`

Completion Message

Payload (version 2.0)	Type	Required
`{`		
`"activeInterface": "consumer"`	string	✓
`}`		

Properties

activeInterface
Specifies the active interface as one of the following values:

- `consumer` - The consumer interface.
- `operator` - The operator interface.

Event Messages

None

24.2.3 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 2.0)	Type	Required
`{`		
`"activeInterface": "consumer"`	string	✓
`}`		

Properties

activeInterface

Specifies the active interface as one of the following values:

- `consumer` - The consumer interface.
- `operator` - The operator interface.

Completion Message

Payload (version 2.0)
`This message does not define any properties.`

Event Messages

None

24.3 Unsolicited Messages

24.3.1 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 2.0)
This message does not define any properties.

24.3.2 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 2.0)	Type	Required
`{`		
`"activeInterface": "consumer"`	string	✓
`}`		

Properties

activeInterface

Specifies the active interface as one of the following values:

- `consumer` - The consumer interface.
- `operator` - The operator interface.

25. 3.x Migration

This chapter provides information on how to migrate to and from XFS 3.x.

Migration to and from 3.x is a key requirement of XFS4IoT. This chapter describes how functionality available in XFS 3.x can be provided and/or derived to and from XFS4IoT.

25.1 CDM (Cash Dispense Module)

The CDM class provides access to cash dispensing functionality. In XFS4IoT, it has been restructured to use the CashDispenser, Storage and CashManagement interfaces.

25.1.1 WFS_INF_CDM_CASH_UNIT_INFO

storage unit storage information is defined by the WFS_INF_CDM_CASH_UNIT_INFO category which is replaced in XFS4IoT by Storage.GetStorage. The following table lists the output fields in WFS_INF_CDM_CASH_UNIT_INFO and how they are mapped in XFS4IoT.

The concept of logical and physical storage units does not exist in XFS4IoT, only physical units are reported. The 3.x logical cash counts can be easily calculated from the given cash counts as described below.

Storage.SetStorage provides all the same functionality as the 3.x command but also provides additional functionality:

1. Only fields which need to be changed need to be provided, including only modifying storage units which are changing.
2. Storage.GetStorage provides a much more detailed breakdown of the capabilities and status of the storage units than the 3.x equivalents, therefore it is possible to decide in advance what a storage unit can be configured to do using *Storage.SetStorage*.
3. *Storage.SetStorage* only accepts as input the fields which can be modified.

3.x	XFS4IoT
WFSCDMCUINFO::usTellerID	Not supported, equivalent to 0 in XFS 3.x
WFSCDMCUINFO::usCount	Number of storage items with a cash interface
WFSCDMCASHUNIT::usNumber	index
WFSCDMCASHUNIT::usType	types. XFS4IoT has the option to support multiple types which is not available in 3.x, but an intelligent map can be performed, for example if the unit supports *cashIn* and *cashOut* then it can be mapped to *WFS_CDM_TYPERECYCLING*.
WFSCDMCASHUNIT::lpszCashUnitName	name
WFSCDMCASHUNIT::cUnitID	id
WFSCDMCASHUNIT::cCurrencyID	currency
WFSCDMCASHUNIT::ulValues	value. An additional conversion may be required due to the different types of these two items. *value* is the absolute value of the cash item, whereas *ulValues* is the value expressed in minimum dispense units. If the *value* of any of the items supported by the device can't be expressed as a C ULONG (integer between 0 and 0xffffffff) then a currency exponent will be required - see CashManagement.GetBankNoteTypes.
WFSCDMCASHUNIT::ulInitialCount	Total number of items in initial

3.x	XFS4IoT
WFSCDMCASHUNIT::ulCount	If not a retract unit, The total count of the items in the unit (*ulCount* in 3.x) is not reported directly but can be derived from the initial, out and in counts. The number of items in the unit is *initial* + *in* - *out*. However for units which dispense items, this calculation should only decremented when the items are either known to be in customer access or successfully rejected, therefore the intermediate *out* fields are not included in this calculation:stacked, transport, unknown and diverted. If counts being incorrectly set at replenishment time means that this would result in a negative number, this should report 0. A dispense storage unit which is not empty but has a derived *ulCount* of 0 may be dispensed from if locally configured to do so - this can cause problems for 3.x applications as *ulCount* can not be negative therefore it can't be used to track dispensed items once that point is reached. XFS4IoT deals with this issue by having separate counts for items *in* and *out*. If a retract unit, retractOperations
WFSCDMCASHUNIT::ulRejectCount	Total number of items in rejected
WFSCDMCASHUNIT::ulMinimum	lowThreshold
WFSCDMCASHUNIT::ulMaximum	highThreshold
WFSCDMCASHUNIT::bAppLock	appLockOut
WFSCDMCASHUNIT::usStatus	If operationStatus reports *dispenseInoperative*, then map to *WFS_CDM_STATCUINOP*, otherwise status if not *ok*, otherwise replenishmentStatus
WFSCDMCASHUNIT::usNumPhysicalCUs	No support for logical units in XFS4IoT. This is equivalent to XFS 3.x value 1.
WFSCDMCASHUNIT::ulDispensedCount	Add the total number of items in all of the out properties.
WFSCDMCASHUNIT::ulPresentedCount	Total number of items in presented
WFSCDMCASHUNIT::ulRetractedCount	Total number of items in retracted
WFSCDMPHCU::lpPhysicalPositionName	positionName
WFSCDMPHCU::cUnitID	id
WFSCDMPHCU::ulInitialCount	Total number of items in initial
WFSCDMPHCU::ulCount	0 if a retract unit which cannot count items during a retract. In all other cases, start with *WFSCDMCASHUNIT::ulCount* and subtract the total number of items in stacked, transport and unknown. If this results in a negative number, this should be 0.
WFSCDMPHCU::ulRejectCount	Total number of items in rejected
WFSCDMPHCU::ulMaximum	capacity
WFSCDMPHCU::usPStatus	If operationStatus reports *dispenseInoperative*, then map to *WFS_CDM_STATCUINOP*, otherwise status if not *ok*, otherwise replenishmentStatus Note this is a change from XFS 3.x where physical status is not overridden by count thresholds (ulMaximum and ulMinimum).
WFSCDMPHCU::bHardwareSensor	hardwareSensors
WFSCDMPHCU::ulDispensedCount	Add the total number of items in all of the out properties.
WFSCDMPHCU::ulPresentedCount	Total number of items in presented

3.x	XFS4IoT
WFSCDMPHCU::ulRetractedCount	Total number of items in retracted

25.2 CIM (Cash-In Module)

The CIM class provides access to cash accepting functionality. In XFS4IoT, it has been restructured to use the CashAcceptor, Storage and CashManagement interfaces.

25.2.1 WFS_INF_CIM_CASH_UNIT_INFO

storage unit storage information is defined by the WFS_INF_CIM_CASH_UNIT_INFO category which is replaced in XFS4IoT by Storage.GetStorage. The following table lists the output fields in WFS_INF_CIM_CASH_UNIT_INFO and how they are mapped in XFS4IoT.

The concept of logical and physical storage units does not exist in XFS4IoT, only physical units are reported. The 3.x logical cash counts can be easily calculated from the given cash counts as described below.

Storage.SetStorage provides all the same functionality as the 3.x command but also provides additional functionality:

1. Only fields which need to be changed need to be provided, including only modifying storage units which are changing.
2. Storage.GetStorage provides a much more detailed breakdown of the capabilities and status of the storage units than the 3.x equivalents, therefore it is possible to decide in advance what a storage unit can be configured to do using *Storage.SetStorage*.
3. *Storage.SetStorage* only accepts as input the fields which can be modified.

3.x	XFS4IoT
WFSCIMCASHINFO::usCount	Number of storage items with a cash interface
WFSCIMCASHIN::usNumber	index
WFSCIMCASHIN::fwType	types. XFS4IoT has the option to support multiple types which is not available in 3.x, but an intelligent map can be performed, for example if the unit supports *cashIn* and *cashOut* then it can be mapped to *WFS_CIM_TYPERECYCLING*.
WFSCIMCASHIN::fwItemType	items
WFSCIMCASHIN::cUnitID	id
WFSCIMCASHIN::cCurrencyID	currency
WFSCIMCASHIN::ulValues	value. An additional conversion may be required due to the different types of these two items. *value* is the absolute value of the cash item, whereas *ulValues* is the value expressed in minimum dispense units. If the *value* of any of the items supported by the device can't be expressed as a C ULONG (integer between 0 and 0xffffffff) then a currency exponent will be required - see CashManagement.GetBankNoteTypes.
WFSCIMCASHIN::ulCashInCount	Add the total number of items in all of the in properties.
WFSCIMCASHIN::ulCount	If not a retract unit, The total count of the items in the unit (*ulCount* in 3.x) is not reported directly but can be derived from the initial, out and in counts. The number of items in the unit is *initial* + *in* - *out*. However for units which dispense items, this calculation should only decremented when the items are either known to be in customer access or successfully rejected, therefore the intermediate *out* fields are not included in this calculation:stacked, transport, unknown and diverted. If counts being incorrectly set at replenishment time means that this would result in a negative number, this should report 0. If a retract unit, retractOperations

3.x	XFS4IoT
WFSCIMCASHIN::ulMaximum	highThreshold
WFSCIMCASHIN::usStatus	If operationStatus reports *depositInoperative*, then map to *WFS_CIM_STATCUINOP*, otherwise status if not *ok*, otherwise replenishmentStatus
WFSCIMCASHIN::bAppLock	appLockIn
WFSCIMCASHIN::lpNoteNumberList	Combination of: 1. Start with initial 2. Add in 3. Remove out
WFSCIMCASHIN::usNumPhysicalCUs	No support for logical units in XFS4IoT. This is equivalent to XFS 3.x value 1.
WFSCIMCASHIN::lpszExtra	Any additional vendor-specific properties included in this storage unit not defined by the schema can be added to this field.
WFSCIMCASHIN::lpusNoteIDs	cashItems lists the cash items which the unit is configured to accept. This can be cross-referenced with CashManagement.GetBankNoteTypes to obtain the noteID for the given cash item.
WFSCIMCASHIN::usCDMType	types. XFS4IoT has the option to support multiple types which is not available in 3.x, but an intelligent map can be performed, for example if the unit supports *cashIn* and *cashOut* then it can be mapped to *WFS_CIM_TYPERECYCLING*.
WFSCIMCASHIN::lpszCashUnitName	name
WFSCIMCASHIN::ulInitialCount	Total number of items in initial
WFSCIMCASHIN::ulDispensedCount	Add the total number of items in all of the out properties.
WFSCIMCASHIN::ulPresentedCount	Total number of items in presented
WFSCIMCASHIN::ulRetractedCount	Total number of items in retracted
WFSCIMCASHIN::ulRejectCount	Total number of items in rejected
WFSCIMCASHIN::ulMinimum	lowThreshold
WFSCIMPHCU::lpPhysicalPositionName	positionName
WFSCIMPHCU::cUnitID	id
WFSCIMPHCU::ulCashInCount	Add the total number of items in all of the in properties.
WFSCIMPHCU::ulCount	0 if a retract unit which cannot count items during a retract. In all other cases, start with *WFSCIMCASHIN::ulCount* and subtract the total number of items in stacked, transport and unknown. If this results in a negative number, this should be 0.
WFSCIMPHCU::ulMaximum	capacity
WFSCIMPHCU::usPStatus	If operationStatus reports *depositInoperative*, then map to *WFS_CIM_STATCUINOP*, otherwise status if not *ok*, otherwise replenishmentStatus Note this is a change from XFS 3.x where physical status is not overridden by count thresholds (ulMaximum and ulMinimum).
WFSCIMPHCU::bHardwareSensors	hardwareSensors
WFSCIMPHCU::lpszExtra	Any additional vendor-specific properties included in this storage unit not defined by the schema can be added to this field.
WFSCIMPHCU::ulInitialCount	Total number of items in initial

3.x	XFS4IoT
WFSCIMPHCU::ulDispensedCount	Add the total number of items in all of the out properties.
WFSCIMPHCU::ulPresentedCount	Total number of items in presented
WFSCIMPHCU::ulRetractedCount	Total number of items in retracted
WFSCIMPHCU::ulRejectCount	Total number of items in rejected

25.2.2 WFS_SRVE_CIM_COUNTACCURACYCHANGED

The count accuracy is reported as part of the Storage for a storage unit, therefore when the count accuracy changes, a Storage.StorageChangedEvent will be generated.