Jernbane-
direktoratet

Guidelines
Handbook V821

Electronic Ticketing

Part 22: National Order Database (Nasjonal ordredatabase)

Dokument nr.: 201800414-11
Dato: 28.02.2018

Foreword

This is a direct continuation of the Handbook V821 previously published by the Norwegian Public Roads Administration. The responsibility of this Handbook was transferred to the Norwegian Railway Directorate on April 1st, 2017.

Handbook V821 concerns electronic ticketing with travelcards, primarily focused on public transportation. The handbook is commissioned by the Ministry of Transportation and is financed by the Norwegian Railway Directorate and the Ministry of Transportation. The main purpose of the handbook is to make it easier for the customer to travel by public transportation. An important part of this simplification is coordination of systems for electronic ticketing on the local, regional and national levels.

The targeted audience for the handbook will be decision makers in public transportation companies and public agencies. In addition, it will address personnel working with requirement specifications and acquisition of systems for electronic ticketing.

A complete overview of the contents in Handbook V821 is given in Part 0.

This is part 22 of Handbook V821. The part gives a description of the National Order Database (NOD) and requirements for clients and systems using it.

The Norwegian Railway Directorate presupposes that current international standards and guidelines given in Handbook V821 Electronic Ticketing are followed by projects for electronic ticketing as instructed by the license authority, ref the Norwegian law for professional transportation (Yrkestransportloven) and the corresponding regulations for professional transportation (Yrkestransportforskriften) §30, which is elaborated by the Ministry of Transportation's circular N-1/2006, and regulations for ticketing in rail transport (Forskrift om billettering ved jernbanetransport) §4.

The Norwegian Railway Directorate, February 2018

Prosjektnummer: 600002	Ansvarlig avdeling: Marked og Samfunn Faglig ansvar: Markedskunnskap
Versjon: 1.0	Forsidefoto/illustrasjon: Statens vegvesen, Skyss, NSB, Ruter, Fjord1
ISBN: 978-82-8386-001-6	

Innhold

1 Introduction

The National Order Database (NOD) is a central component for supporting online electronic ticketing and Internet sales. Instead of distributing action lists to offline equipment, all orders are stored in a common order database. When a card is presented a device (client) supporting the NOD, the NOD client will call an online interface to retrieve the orders for that specific card in real time. These clients may be integrated in existing hardware or established as pick-up devices (PUD) dedicated for this purpose. The NOD clients will be controlled on a low level by the NOD when performing orders.

An order may be issued targeted for either a contactless card, a mobile telephone client or a device. In the future also other types of clients may be added.

This document describes how the NOD functions and how it communicates with other parties.

1.1 Logical Architecture

The logical architecture of the NOD solution is shown in fig 1.1. This shows an example using an Internet ticket sales order to be issued on an NSD contactless smartcard.

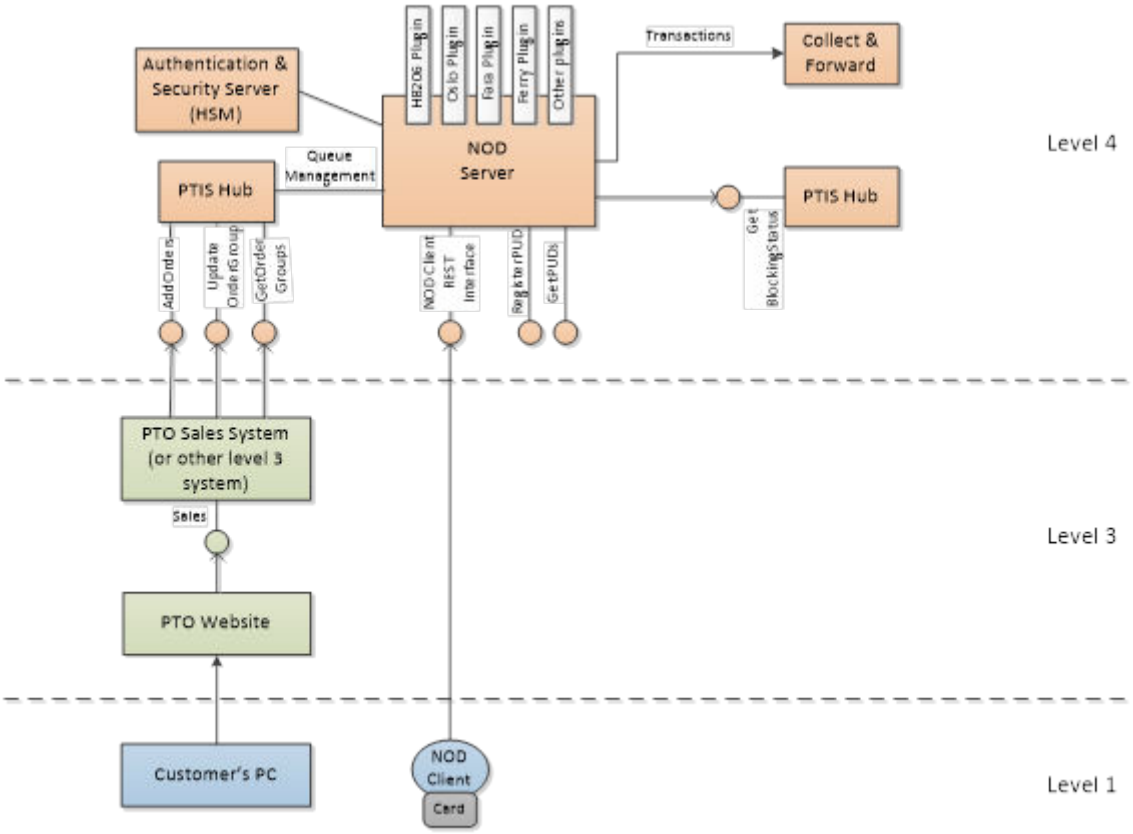

Figure 1-1: NOD Logical Architecture

Normally it will be the PTOs' sales systems that issues orders to the NOD Server via the PL4 system in the Public Transport Information and Service Hub (PTIS Hub). This process and its requirements and interfaces are described in detail in chapter 3.

After receiving an order, the NOD Server stores the order and waits for a request for the corresponding card. When the card is presented on a pick-up device, the device sends the card id to the NOD Server. The detailed requirements and interfaces for the NOD Client are described in chapter 4. The NOD Server then retrieves the complete binary card contents from the PUD, and gives it to the correct plugin together with the order details.

The plugin performs the necessary card level logic and returns a new binary card image and contents for the transaction. The detailed requirements and interfaces for the plugins are described in chapter 5. The NOD Server then compares the old and new card image using a diff engine, and based on this creates the corresponding APDU (Application Protocol Data Unit) level commands for the PUD. The PUD then retrieves the commands through a REST interface, leading it step by step through the whole process of writing to the card. The NOD Server also decides when authentications are necessary and which keys to authenticate with. Whenever an authentication is required towards the card, the crypto operations is handled by using the authentication and security server.

When the card writing process is finished, the plugin generates the corresponding transactions on behalf of the PUD, and sends these to the Collect and Forward central. The NOD Server itself is described in chapter 2.

1.2 Order Types

There will be different types of orders available. Not all order types will be available in phase 1. The sequence they will be implemented in depends on the needs of PTOs.

- Stored value reload
- Product sale
- Auto renew management
- Auto reload management
- Product deletion
- Unblock card or product
- Validation
- Personalization
- Refund (for sales offices)
- Cancelation (for sales offices)
- Card/application issuing (for sales offices)
- Reconstruction
- Garbage collection
- Perform offline action lists
- Support for ticketing on mobile telephones using 2D barcodes (QR codes)

When issuing products only NSD (MIFARE DESFire) will be supported in phase 1. In the future also Ultralight, paper and possibly other ticket media will be supported as well.

2 The National Order Database Manager

2.1 Order Management Process

The NOD Server uses services in the Public Transport Information and Service Hub (PTIS Hub) towards the PTO clients. These will exist side by side with other services, e.g. private action lists and blocking lists.

The orders will be stored there and immediately, put on an execution queue and forwarded to a separate database for active orders.

2.1.1 State Engine

The state engine describes the different states an order may have, and legal transitions between them. The state chart is shown in fig 2.1.

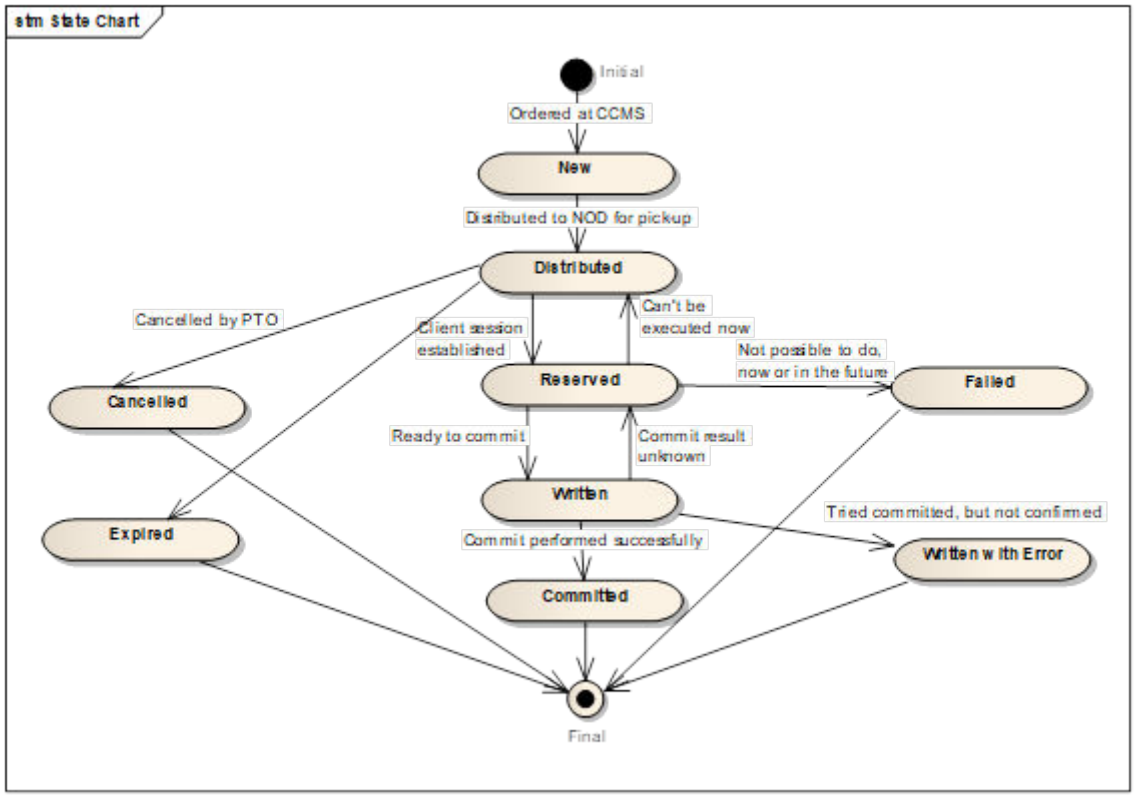

Figure 2-1: Order state chart

An order is set to New when ordered by a PTO at the PTIS Hub AddOrder service. It will almost immediately be put on the NOD queue and become Distributed. As long as it has status Distributed it may be cancelled by the PTO through the UpdateOrderGroup service.

When a NOD client is presented with e.g. the card the order is issued to, it retrieves all order groups with status Distributed, set them to Reserved, and pass them to the correct plugin(s). The plugin decides if the order may be executed. If it may not be executed at that time, but maybe in the future, the state is returned to Distributed, thus allowing it to be retried the next time the card is presented.

If an order cannot be executed, now or in the future, the state will be set to Failed. This will prevent any future attempts to execute the order.

If it may be executed now, the plugin returns the updated image, which is passed to the Diff Engine. The Diff Engine creates the APDU (or equivalent) commands to the ticket medium. The APDU commands are put together in command sets, based on necessary authentications. For one order group, only one commit shall be written. Therefore, only one record may be added to any record file, thus eliminating the possibility for partially executed orders.

When the command set with the Commit command has been sent to the client, the state is set to Written. If the client returns OK, the state is set to Committed, and the order execution is complete. If no answer is received, the order will return to Reserved state. The next time the card is presented, it will be checked to see if the previous attempt to commit was successful or not. This is done by storing a checksum of the card image as it was when it was first presented and the checksum of the expected outcome after commit (as created by the plugin). When the card is presented anew, the checksum of the current card image will be compared to the stored checksums. If it is the same as the initial, it means the commit failed, and the anti-tear mechanisms rolled back the changes. The order will then be executed again. If the checksum is the same as the result from the plugin, it means that the commit was successful, and the state is set to Committed. If it has any other value this means that something else has happened in between. The card then needs to be inspected manually and the order state set accordingly afterwards.

Note that the NOD server is not tracking individual Order statuses, but the status of an entire Order Group. This individual tracking is not possible as execution of several orders may be merged into a few Client Commands.

The results of individual Nod Client Commands or Plugin responses may however accompany a Group status update to facilitate diagnosis of an execution failure.

2.1.2 Order Group Status Lifecycle

NEW

When a new ordergroup is added to PL4 it will initially have the NEW status.

State Transition	Event Description
DISTRIBUTED	This state will transition to the DISTRIBUTED state immediately upon submission to PL4, as all Order Groups currently are submitted to the NOD immediately.

There is currently no support for delayed distribution to the NOD server. Any delays must be implemented on the PTO side.

DISTRIBUTED

When the Order Group is added to the NOD server queue it will have the DISTRIBUTED state. The state may be set to DISTRIBUTED from RESERVED if the order could not be executed at this time. This enables another attempt to be made to execute the Order Group.

This is the only state from which it is possible to cancel the distribution.

State Transition	Event Description
RESERVED	A NOD Client creates a NOD session for the purpose of executing this group.
CANCELLED	A PTO Asks PL4 to cancel the distribution.
EXPIRED	The Order Group distribution has passed the ExpiredDate of the Order Group.
FAILED	For a permanent reason the Order Group should not be distributed any more.

RESERVED

When a NOD Client successfully establishes a NOD Session for an Order Group, the RESERVED status will be set.

State Transition	Event Description
WRITTEN	The Commit client command is submitted to the NOD Client, no result is returned yet.
DISTRIBUTED	For a temporary reason, the NOD Client or Plugin could not complete the Order Group and eventual changes are rolled back.
FAILED	For a permanent reason the Order Group should not be distributed any more.

WRITTEN

When the command sets executing the Order Group has been sent to the NOD Client the WRITTEN status will be set.

State Transition	Event Description
COMMITTED	The Commit client command is submitted to the NOD Client, and the result is confirmed by an incoming commit result, or deduced by a comparison of the incoming source image with the target image from the previous NOD Session.

WRITTEN WITH ERROR	The Commit client command is submitted to the NOD Client, but the result could not be verified.
RESERVED	A NOD Client retries while the previous NOD Session is still active.If the source image is unchanged from the previous source image, continue as normal.
FAILED	For a permanent reason the Order Group should not be distributed any more.

Note that comparisons with previous source and target images are only relevant when a Nod Client/user aborts the dialogue with the NOD server, and then immediately retries before the NOD Session times out.

COMMITTED

When a NOD server receives a confirmation that the commit has been successfully executed by the NOD Client the COMMITTED status will be set.

The Order Group is removed from the NOD by a batch job.

This is an end state.

WRITTEN WITH ERROR

If the NOD server does not receive a confirmation of successful execution of a command set from the NOD Client before the NOD Session times out, the WRITTEN WITH ERROR status will be set.

The Order Group is removed from the NOD by a batch job.

This is an end state.

CANCELLED

When a PTO cancels an order in the DISTRIBUTED state using the PL4 order interface the CANCELLED status will be set.

The Order Group is removed from the NOD by a batch job.

This is an end state.

FAILED

If the NOD server determines that the order may not be executed now or in the future the FAILED status will be set.

The Order Group is removed from the NOD by a batch job.

This is an end state.

EXPIRED

The NOD server periodically checks that the ExpiredDate of an Order Group is still valid for distribution.

The Order Group is removed from the NOD by a batch job.

SYSTEM_ERROR

If an unexpected system error happens during any stage of the Order Group lifecycle, the Order Group is set directly to this state. The state is used to move an Order Group out of distribution until the cause has been identified and corrected, so that the Order Group do not block subsequent Order Groups.

The incident will have to be investigated by the System Administrator or technicians, and the state will have to be changed manually according to the investigation result in both NOD and PL4.

The Order Group is NOT removed from the NOD by a batch job, and may therefore be re-enabled.

State Transition	Event Description
WRITTEN WITH ERROR	Manual investigation confirmed that the Order Group possibly has been committed before failure
FAILED	Manual investigation confirmed that the Order Group should be permanently failed.
CANCELLED	The Order Group may be cancelled via PL4, so that the traveller may get a refund.
DISTRIBUTED	The Order Group may be re-distributed via PL4, so that the traveller may attempt to pick up the Order Group once more. This should normally not be done unless the error has been identified and fixed.

This is an end state.

2.1.3 Order Group Status Codes

The following status codes are defined:

```
NEW: 0
DISTRIBUTED: 1
RESERVED: 2
WRITTEN: 3
COMMITTED: 4
WRITTEN WITH ERROR: 5
CANCELLED: 6
FAILED: 7
EXPIRED: 8
SYSTEM_ERROR:9
```

2.1.4 Order Group Status Code Visibility

The status codes are used to indicate Order Group status in PL4 and in the NOD. The table below indicates which states are visible in PL4 and which states are visible in NOD. By temporary we mean that the existence if this status is subject to the PL4 and NOD server implementation, in particular the batch job scheduling. Until the scheduler has executed, the status may temporarily be visible.

	NEW	DISTRIBUTED	RESERVED	WRITTEN	COMMITTED	WRITTEN WITH ERROR	CANCELLED	FAILED	EXPIRED	SYSTEM_ERROR
PL4	(temporary)	X			X	X	X	X	X	X
NOD		X	X	X	(temporary)	(temporary)	(temporary)	(temporary)	(temporary)	X

2.1.5 Batch Jobs

Nod Session Timeout

A job is periodically releasing NOD Sessions that have not been completed by the NOD Client for any reason. The period between scheduling of this job is defined as the NOD Session Timeout. The NOD Session Timeout is a NOD Server system configuration, but may be expected to lie between 20-30 seconds.

All Groups that have status RESERVED are released from the NOD Session. This releases the Groups for NOD Distribution to other clients.

All Groups that have status WRITTEN picks up eventual transactions generated by the plugins and submits this to PL4 with Group status code WRITTEN WITH ERROR, in addition to an error description. This removes the Group from NOD distribution. The transactions themselves will also set the error flag in the transaction header.

The NOD session itself is removed and further queries by the NOD Client will receive 404 NOT FOUND on URL's referring to the session.

Expiration

Each Order Group is registered with an ExpirationDate that indicates when the Order Group should be removed from distribution.

All Groups that have ExpirationDate older than current NOD System time will be set to EXPIRED. This removes the Groups from NOD distribution.

The order group in PL4 is modified with the status change EXPIRED.

Garbage collection

Order groups that have reached a final state is no longer relevant for NOD distribution. The NOD may however wish to keep these transactions for a period of time to facilitate customer support inquiries or debugging. The removal of this data is therefore done by a customizable batch job.

All order groups with status in (COMMITTED, WRITTEN WITH ERROR,CANCELLED,FAILED, EXPIRED) will be deleted or moved into separate storage.

2.1.6 PL4 Batch Jobs

Transaction distribution

PL4 will continually receive transactions from the NOD that are generated by the plugins.

When a certain amount of transactions are received or a specified time interval has passed, PL4 will generate IOS-compatible XML export files and store these on a filesystem that is synchronized with the import-folder of the IOS system.

A scheduler will check if there are transactions received from NOD to be distributed to IOS. No more than 10 000 transactions will be distributed per file.

Only transactions belonging to Order Groups with status COMMITTED or WRITTEN WITH ERROR should be distributed.

To decide which transactions have been distributed a fileID is generated. This fileID is updated on every row ready to be distributed. This will prevent other incoming transactions to be added to the file.

The file generated is a xdr file on the same format that PL4 imports from IOS.

2.2 Difference Engine

The implementation of a difference-engine is not part of the NOD specification, but is described here to give a better understanding of how the NOD works.

The difference engine in its most basic form identifies the differences between the card image delivered by the NOD client and the target image calculated by the plug-in. The main purpose for the diff-engine is to communicate this difference to the medium such as a DESFIRE Card.

Since the difference engine works with images, this allows several plugins to be chained, and a single difference can be calculated across the work of several plugins.

This means that there will be developed one difference engine for the namespace http://ioas.no/nod/client/commands/desfire/apdu that generates the NOD Client dialogue containing the APDU commands needed to realize the difference in the DESFIRE images with APDU commands. Another difference engine may be developed to realize differences on Ultralite cards and images with commands that belongs to this namespace.

Which diff-engine will be chosen depends on the capabilities that the NOD Client supports and the physical media presented to it.

2.2.1 Command Decorators

In addition to realizing a change on the target medium, a difference engine will also communicate with the NOD Client itself, for example by blinking lights, sound etc. These commands are generated by the Command Decorators, one decorator for each namespace the NOD Client supports. These decorators have the opportunity to inject commands into the NOD Client dialogue when specific difference engine events occurs such as start, commit, failure etc.

For example, if a specific NOD Client supports writing a receipt to paper, a Command Decorator will be attached to the difference engine that inserts a command to the printer in its own custom namespace at the difference engine commit event. This decorator will only be attached to the difference engine if the NOD Client declares that it supports this through the capability declaration.

Decorators and Diff-engines can be mixed; a buzzer decorator could be used both by the Desfire diff-engine and the Ultralite diff-engine.

2.3 Transaction Generation

All plugins may generate transactions according to the DIS specification.

When a card is presented to the NOD Client, the card image is retrieved from the card together with the NOD Client context parameters. This is passed to the plug-in together with the order description.

When the plugin has modified the source image according to the order, the target image must be returned together with eventual transactions according to the DIS specification. The rules for how

these transactions are generated are the same as described in HB V821 part 19. Orders shall be managed in the same manner as actions.

The Plugin Interface specification has moved the transaction details into a separate namespace to allow for future changes in accordance to DIS changes without breaking the NOD Plugin interface. This means that different plugins may support different versions of the DIS specification in transitional periods, as long as IOS still supports importing transactions from the given DIS version.

The generated transaction will be submitted to the Collect & Forward system (IOS) by the NOD Server when the Order Group status transitions to COMMITTED, FAILED or WRITTEN WITH ERROR. The NOD server will set the transaction attribute TransactionHeader.TransactionStatus accordingly.

3 Requirements for Client Sales Systems

3.1 Issuing Orders

New orders are generated by calling the AddOrder service. In addition to the order data, the ordering company must provide its internal order reference. This reference is the same as used for actions, and must be unique between both orders and actions. The order reference will be reported as part of the ActionID in the resulting transactions. The order contents will be checked towards the corresponding XSD. If it passes it will be forwarded to the NOD Server. The NOD Manger does not have any functionality in regard of payment handling. All payments must therefore be handled by the ordering PTO before the AddOrder service is called.

3.2 Managing Orders

After issuing an Order by creating an Order Group, the Order group can only be cancelled. This is done by updating the Order Group in PL4 to status CANCELLED. All refunds and customer management due to this is the PTO's responsibility.

3.3 Webservice Order Interface Definitions

The following webservice interfaces are implemented in the PTIS Hub to support NOD Orders:

- AddOrders
- GetOrderGroups
- UpdateOrderGroup

See App D for a detailed description of the different services

3.3.1 AddOrders

The AddOrders interface allows a PTO to add a NOD Order. The Order will then be distributed to the NOD Server and made available to NOD Clients. XML Schema validation is performed on the input. PL4 will perform a mapping operation that will decide which NOD Server Plugin should process the order, this decision is based on the input data.

The request and response xml is specified by the AddOrdersRequest and AddOrdersResponse elements in the OrderServices.xsd XML Schema.

3.3.2 GetOrderGroups

The GetOrderGroups interface allows a PTO to retrieve a list of order groups based on a set of search criteria. Note that PTIS Hub will also have a configurable maximum value that regulates the amount of order groups the client can ask for.

The request and response xml is specified by the GetOrderGroupsRequest and GetOrderGroupsResponse elements in the OrderServices.xsd XML Schema.

3.3.3 UpdateOrderGroup

The UpdateOrderGroup interface allows a PTO to set the order group status to "Cancelled".

The request and response xml is specified by the updateOrderGroupRequest and updateOrderGroupResponse elements in the OrderServices.xsd XML Schema.

4 Requirements for NOD Clients

The characteristics of NOD client may vary as NOD Clients will be embedded in hardware with variyng capabilities. The most common NOD clients will be the following:

- Stand-alone pick-up device (PUD)
    - These devices should as a minimum have two LEDs, red and green, a 2x16 characters display and a speaker.
- Integrated client in validators
    - It is not recommended to integrate the NOD client in validators due to latency reasons. Since all presented cards must be looked up online, this will add substantially to the validation time and should only be used when time is not critical. If a NOD client is integrated in a validator, the result of the order will normally be shown at the same time as validation information. In such a case only 16 characters is available for order feedback, for a complete session. If several order groups have been performed, still only one line of 16 characters may be displayed.
- Integrated client in ticket vending machines (TVM)
    - When integrating the NOD client in a TVM it can be done either by implementing it as a separate button/menu choice or by looking up all cards online. Which is most suitable may vary.
- Integrated client in driver's consoles
    - When integrating the NOD client in a driver's console it should be a separate button/menu choice for this. It is not recommended to look up all cards online. It should only be done on the customer's request.
- Integrated in mobile telephones
- Integrated client in manned sales terminals
    - When integrating the NOD client in a sales terminal the sales terminal must on itself find the relevant ticket media ID (card number) and use this when issuing the order. Then it must use the same ID to retrieve the orders from the NOD Server.

4.1 The Order Delivery Process

When a card is presented the NOD client is responsible for the card activation, including the full anti-collision loop (complete chip id is required) and initialization commands (RATS and PSS). The NOD Client shall select the CardIssuer_DF and read the cardNumber32Bits, sending it to the NOD Server. Afterwards the master DF shall be selected. The command sequence from the NOD Server requires the card to be selected and activated, standing in the master DF (root) and not authenticated.

After posting the card id to the NOD Server through the REST interface (see App A and B for detailed interface description) the NOD Server will return available order groups. These will be ordered by registration date. The NOD client shall always perform the orders in the sequence given by the NOD Server, by posting a request to the interface.

4.2 Requirements for the User Interface

All NOD clients must fulfil the following minimum requirements:

1. Text feedback capability, minimum 16 character text display, preferably 2x 16 characters or larger displays/screens. If only 1x or 2x text display is available, also red and green LEDs shall be implemented. Yellow LED may be implemented, but will not be used by NOD Server. For other screen sizes LEDs are optional.

2. Speaker solution, able to either play WAV (preferred) files or to receive playback command sets consisting of one or more frequency (Hertz)/duration (milliseconds) instructions. WAV files will be sent by the NOD Server, but may be cached locally. The same file name will never be used for different sounds.
3. Easily recognizable RF antenna area.

4.3 NOD Client Interface

The NOD client interface is exposed as a REST interface (Representational State Transfer), using the HTTPS protocol. The REST interface builds on the NOD common REST interface specification. This is further described in app A. The main services are shown in ch 4-2.3.

4.3.1 Capabilities

Different NOD Clients will have different capabilities in regards of which orders that it is possible to perform:

- Supported media (NSD (DESFire), Ultralight, 2D barcode, paper etc)
- Communication (GPRS, Edge, 3G, LAN etc)
- Type of terminal (self-serviced, sales office, validator, dedicated PUD, mobile telephone etc)
- Terminal specification
    - Screen
        - Full PC screen (TVM, sales office), may show detailed info for each order in several groups.
        - Small LCD (color or B/W), only show summary info for each order group.
        - Text display (1x16 or 2x16), only show summary info for all order groups.
    - LED (red, yellow, green)
    - Sound
        - WAV capability
        - Simple speaker

The capability code is a declaration of the physical and logical capabilities of the NOD Client. This declaration is used for the following purposes:

- Filtering: Only distribute Order Groups that are relevant for the NOD Client (e.g don't distribute Groups requiring specific hardware to clients that don't have it).
- Choice of Difference Engine: Choose a difference engine that produces commands that can be written to the card with the given NOD Client.
- Command Decorators: Only introduce Commands that communicates with the NOD Client Equipment from namespaces the NOD Client explicitly supports. (e.g don't generate BUZZER commands for units with no speakers).
- Optimization: Provide optimization hints that the NOD server may use to streamline the response.

For more information on the filtering, see ch 5.2 regarding mapping of orders to plugins.

The list of capability codes is described in App B.

Other capabilities may be defined in the future. The Registrar is responsible for maintaining a complete list of capabilities used for the capability flags in the REST interface.

4.3.2 Interface Context Parameters

The context represents context-relative information that the NOD Client shall provide to the NOD Server and Plugin to enable processing of an Order Group. The context contains dynamic values that

the NOD server cannot derive from static sources. One such example is the physical location of a NOD Client that is located on a bus, this may needed for a plugin to calculate zone-related tariffs.

The context values will primarily be forwarded to the NOD Plugin that is processing the Order, so the list of available context properties will grow over time as plugins are developed.

The list of context parameters is described in App B.

User language is also part of the NOD Client context but shall be passed as part of the HTTP Header information. The user language is the chosen user language of the terminal. Some types of terminals, e.g. TVMs support several languages. When such terminal are used as NOD clients the messages from the NOD will as far as possible be in the same language as the user have chosen. NOD will initially only support Norwegian, but will be expanded with English in the future. Phrases not available in the chosen language will be returned in the next language in the priority sequence.

4.3.3 REST Interface

The order retrieval interface on the NOD Server uses a REST interface, based on the standard HTTP 1.1 protocol. Which order groups that are returned to the client are filtered based on the client's capabilities. The returned order groups shall be executed in the sequence returned by the NOD Server.

The NOD Client commands are returned from the NOD whenever the NOD Client should execute commands on the card. The commands are scoped to different namespaces according to the following:

- The ticket medium presented (DESFire, Ultralight, Mobile etc)
- The capabilities of the NOD Client (support of sound, retains card physically etc)
- The version of the NOD Client API the Client supports.

Ideally, no NOD Client should receive a response with an unsupported namespace, as orders requiring the namespace should be filtered out already in the GET /groups REST call according to the NOD Client Capabilities.

The detailed REST interface specification is given in appendix A and B.

4.4 Security

For all NOD Clients basic HTTP authentication and HTTPS will be required.

5 Requirements for NOD Plugins

5.1 Business Logic

Plugins can be used to implement various types of functionality. The first to be implemented will be the HB V821 plugin for managing NSD travel cards. The plugin will then receive the incoming ticket medium image, order group description and context information from the NOD client. Only one order group may be processed at a time. Based on this, the plugin is responsible for returning an updated ticket medium image, corresponding transaction objects and user information to be displayed by the NOD client. When updating the image, the same rules apply as for other ticketing equipment. For NSD this is described in HB V821 part 18.

Other relevant plugins may be for implementing 2D barcodes, MIFARE Ultralight or support for NFC telephones.

The user information must be adapted to the NOD client capabilities in regard of size and complexity.

5.2 Mapping of Orders to Plugins

The mapping is not strictly an interface specification, as the management of mappings is not exposed as services. The mapping procedure is however needed to understand how orders are connected with their respective plugins, and it indirectly describes the minimum functional granularity of a plugin: A plugin cannot support orders more specialized than what can be mapped uniquely by PL4.

When an Order is submitted to PL4, a decision has to be made which plugin the Order should be processed by. PL4 should not have to inspect the OrderDescription itself, as this will require PL4 to be able to parse the description.

A mapping happens for each order in the order group, this means that orders in one order group can be executed by different plugins, in the strict sequence that the orders were added to the group.

Some of the scenarios that are enabled by mapping are:

- When a new version of a plugin supporting new products is rolled out, existing undelivered Orders should still be processed by the previous plugin while the new plugin should be in effect for new Orders at a specific date.
- Different versions of a plugin may also require different versions of the validating schema.
- Different PTOs may want the same product to be processed by different plugins.
- A new plugin only should be operative in a specific test network-ID.
- Many Orders are common, new plugins should only have to implement the missing functionality while existing "basic" plugins take the rest.
- An Order should be processed by a plugin, but should not be distributed to all Nod Clients, as some are older models unable to process the commands. (For example, the Client does not support the physical card required for this order)

This requires a flexible and configurable mapping of an incoming Order against a specific plugin.

The following fields must always be submitted to PL4 in addition to the OrderDescription:

- Action Type
- Company ID

- Network ID
- Template ID
- Purchase dateAdditional Required Capabilities of the NOD Client, these are optional additional restrictions specified by the PTO at submission.

All incoming orders will be matched against a mapping table that allows wildcards for specific values.

5.2.1 OrderMapping Example

Action Type	Company ID	Network ID	Template ID	From Date	To Date	Minimum Capability Requirement	Plugin URI Example
SVRACLedREC	*	*	*	01.01.2012	*	00000000000000001	http://localhost:8080/nod_hb206_plugin/services/2
PSAACLedREC	3	578000	*	*	31.12.2013	00000000000000001	http://localhost:8080/nod_hb206_plugin/services/2
MobileTicket	*	*	*	*	31.12.2012	000000001	http://localhost:8080/nod_mobile_plugin/services/2
*	3	<TEST NETW>	*	*	*	101101101011	http://localhost:8080/nod_hb206_plugin/services/3

The following rules apply:

If an Order matches both a wildcard and a specific value, the most specific value mapping is chosen (the one with the fewest wildcards).

If an Order matches two mappings with the same amount of wildcards, an error is returned

The validation schema for the order description is retrieved by PL4 from the plugin itself on the URL <Plugin URI>/resources/orderSchema.xsd

When an order matches a single mapping, the Order Description is validated against the Plugin Schema, and submitted to the NOD for distribution by the matching plugin.

The incoming Additional capabilities of the NOD Client are merged with the Minimum Capability Requirements of the mapping, and the sum is the required NOD Client capabilities submitted to the NOD.

5.3 Interface

The complete, detailed interface between the NOD Server and the plugin is described in App C.

5.4 Binary Ticket Medium Image Structure

The binary ticket medium image structure will be different for different ticket mediums. In phase 1 only MIFARE DESFire will be supported. The XSD for this object is described in NODPluginImgDESFire.xsd.

5.5 Transaction Requirements

The transactions returned from the plugin must be defined according to HB V821 part 19 and the corresponding ticket medium. An updated XSD for each purpose will be made available from the Registrar.

Appendixes

Appendix A: Common REST Interface Specification

Appendix B: NOD Client REST API

Appendix C: NOD Plugin Interface Specification

Appendix D: NOD Webservices Specification