Statens vegvesen

Electronic Ticketing

Part 22 National Order Database
(Nasjonal ordredatabase)

VEILEDNING Håndbok 206

Vegdirektoratet 2012

Electronic Ticketing

Part 22

National Order Database

Handbooks in the Norwegian Public Road Administration

This is a handbook in the Norwegian Public Roads Administration's (NPRA) handbook series, a collection of consecutive publications primarily intended for use within the agency.

NPRA has the main responsibility for authoring and maintenance of the handbooks.

This handbook is only published on www.vegvesen.no.

The NPRA's handbooks are published on two levels:

Level 1 – Yellow color on the front page – includes regulations, norms and guidelines approved by superior authority or by the NPRA by proxy.
Level 2 – Blue color on the front page – includes instructions, text books and road data approved by the department authorized for this in the NPRA.

Electronic Ticketing

Nr. 206 Part 22 in the Norwegian Public Roads Administration's handbook series.

Responsible department: Traffic Management

ISBN 978-82-7207-639-8

Revision History

Version	Date	Authors	Main changes
1	2012.11.05	Kjell-Erik B. Eilertsen, Karl Ivar Dahl, Kim Richard Hansen	First version of the document.

Foreword

Handbook 206 concerns electronic ticketing, primarily focused on public transportation. The handbook is commissioned by the Ministry of Transportation and is financed by the Norwegian Public Roads Administration 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 206 is given in Part 0.

This is part 22 of Handbook 206. The part gives a description of the National Order Database (NOD) and requirements for clients and systems using it.

The Norwegian Public Roads Administration presupposes that current international standards and guidelines given in Handbook 206 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.

Norwegian Public Roads Administration, July 2012

Table of Contents

List of Figures

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 (as described in HB206 part 21), 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.

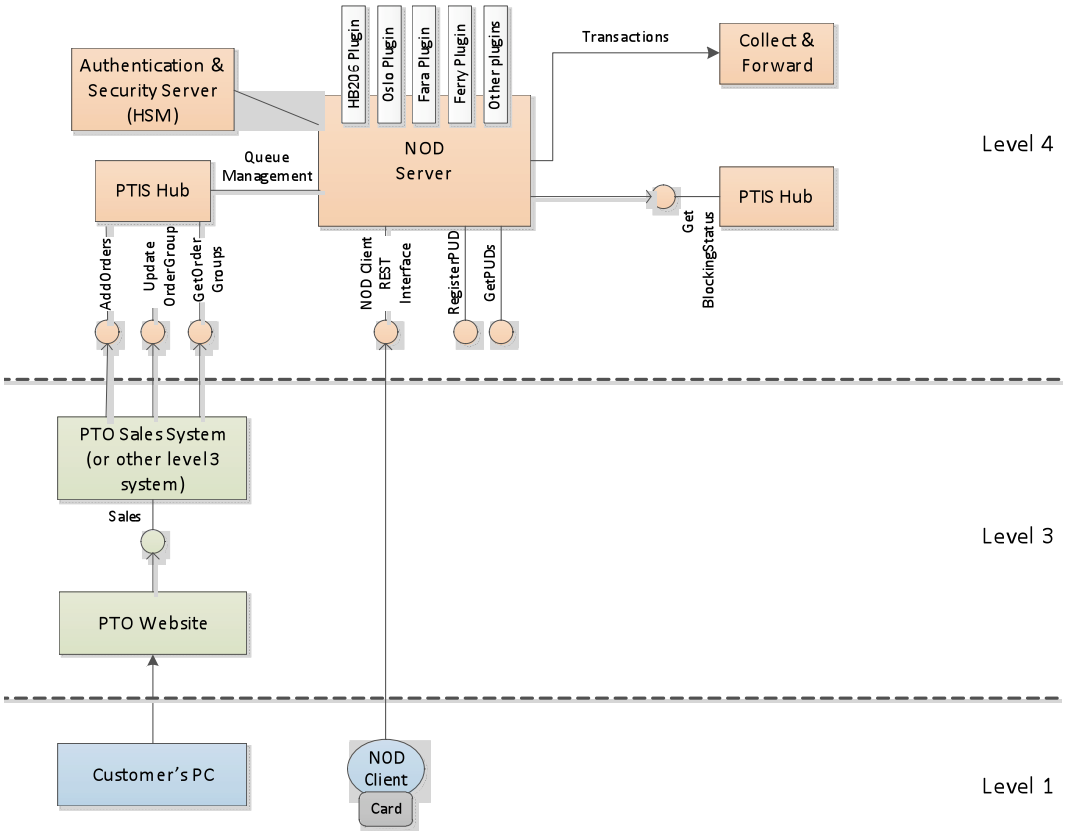

Fig 1.1: NOD Logical Architecture

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.

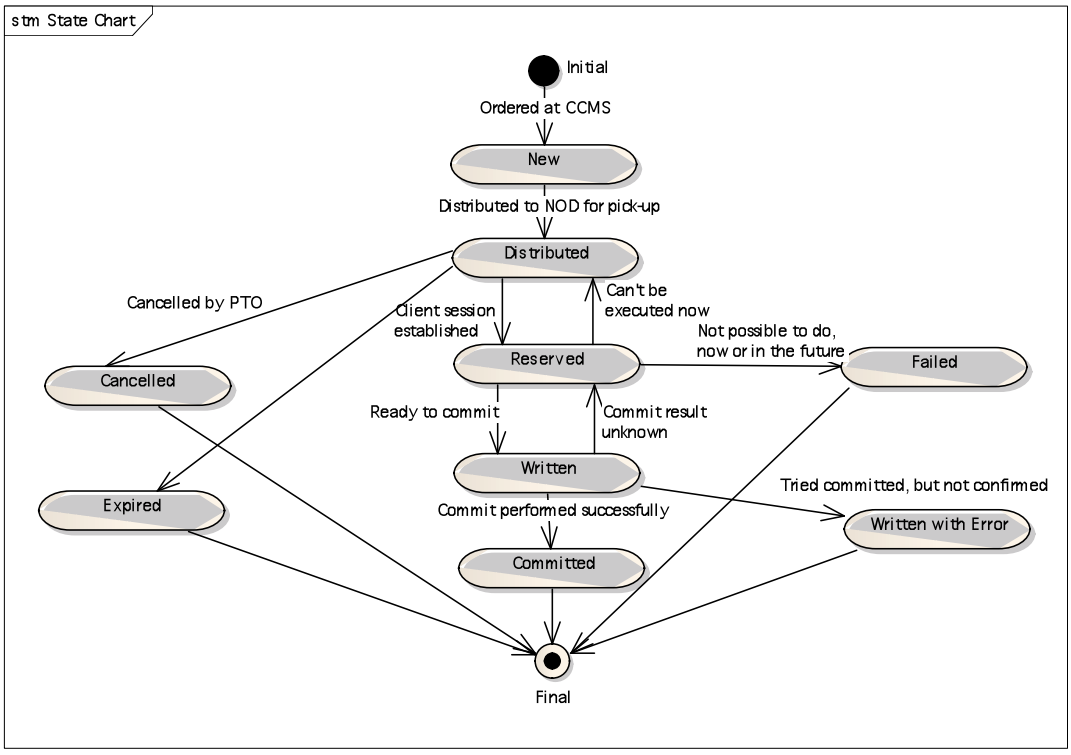

Fig 2.1: Order state chart

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 retrieved 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.

2.1.1 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.

2.1.2 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

2.1.3 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

2.1.4 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.

2.1.5 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 HB206 part 19. Orders shall be managed in the same manner as actions.

The Plugin Interface specification has moved the transaction details into a separate name-space 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 varying 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 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
  - o 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 HB206 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 HB206 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	*	0000000000000001	http://localhost:8080/nod_hb206_plugin/services/2
PSAACLedREC	3	578000	*	*	31.12.2013	0000000000000001	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 HB206 part 19 and the corresponding ticket medium. An updated XSD for each purpose will be made available from the Registrar.

Appendices

Appendix A: Common REST Interface Specification
Appendix B: NOD Client REST API
Appendix C: NOD Plugin Interface Specification
Appendix D: NOD Webservices Specification
Appendix E: NOD Feedback
Appendix F: Static Order Group staticDesfireContents

Appendix A: Common REST Interface Specification

Description

This document specifies the common parts of the interface that all REST APIs will follow. The following specification applies for all NOD REST services unless explicitly declared otherwise.

HTTP-Version/1.1 Protocol Parameters

The NOD server is designed to support HTTP content negotiation. This has numerous benefits such as backwards compatibility to existing clients during changes as well as protocol-support for future content types in future NOD clients. There is initially little to negotiate, but it is important the NOD Clients are designed with this negotiation in mind as support for other content will be added (and possibly be removed) over time.

Common supported Request Headers

The default values specifies what the Server assumes if the header is not provided.

Header	Default	Current Alternatives	Description
Accept	*/*	text/plain, application/xml	The requested encoding of the response data. NOD currently supports text/plain for simple services and application/xml for xml content. In the future this may be expanded with support for binary encodings such as application/fastinfoset or application/exi.
Accept-Charset	utf-8	None	The requested character set for the response text. All text-based requests and responses to NOD should be encoded in UTF-8.
Accept-Language	no	None	The requested language used in feedback messages, dictionary entries etc.
Connection	Keep-Alive	close	Specifies that the physical connection should be kept open for subsequent requests. Keep-Alive is default in HTTP 1.1, so only add this header when the client knows that no further communication is required. It is normally the NOD server that uses this header in the final response to the NOD Client to terminate the SSL connection. The actual Keep-Alive duration is a Server-side configuration setting, subject to performance tuning.
Authorization		base64(NodClientID:Password)	All NOD clients must specify their Client IDs and Password as BASIC authentication for all requests in accordance with RFC2617.
X-NODClient-Capabilities	0		A string representing the capabilities of the NOD Client. This header must be present in all requests to the NOD Server that adapts to Client Capabilities

The NOD will over time support other header value alternatives and the Client should ask for the preferred values with the default last. For example; A Nod Client on a device that allows language preferences should signal this by sending Accept-Language: en;q=1, no;q=0.9 In this case English feedback messages will be returned in the future when the NOD supports it. The actual language of returned content will be specified in the Content-Language response attribute.

Common Response Headers

The default values specifies what the the Client should assume if the header is not provided.

Header	Default	Current Alternatives	Description
Content-Type	text/plain;charset=utf-8	application/xml	The actual content datatype of the response data.
Content-Language	no	None	The actual content language of the response data.
Retry-After		Any number of seconds	This field may be returned on any response in connection with HTTP Errorcode 503 (Service Unavailable). The client MUST wait at least the specified number of seconds before resubmit to avoid overloading the server.
Connection	Keep-Alive	close	Specifies that the physical connection should be kept open for subsequent requests. Keep-Alive is default in HTTP 1.1, so only specify close when the server knows that no further communication is required.

Common Service Parameters

Most data is encoded according to the associated XML Schema specification of the data structure the REST service uses. There are however a number of re-occurring parameters that are submitted as request parameters.

Note that the parameter datatype is String, even when they initially only contain integers. This is to enable future flexibility and scalability. For example, a numeric sequence is a bottleneck in a highly scalable system. Therefore the NOD Client should not assume these values to be integers, in sequence, or numeric at all. This means that a Command Set ID might be 1-10 today, but may change to UIDs in the future. A MediaSerialNumberID may be numeric today, but may also be strings to identify new types of media that use a different kind of identifier.

ParamName	Type	Description
mediaSerialNumberID	String	The identification ID of the physical media presented to the NOD Client, typically the MediaSerialNumber of a DESFIRE Card. Treated as string to enable future flexibility.
capabilityCode	String	A code describing provided capabilities or required capabilities. See separate description of this parameter.
nodSessionID	String	A generated ID representing a globally unique NOD Session.
groupID	String	A generated ID uniquely representing a group of Orders that should be executed as one transaction.
commandSetID	String	A generated ID representing a specific Command Set in the dialogue between the NOD Client and Server, unique relative to the groupID.

HTTP Return codes

These are generic responses, when the result code should be understood in the context of a given service, the error code is described in the relevant service specification.

Code	Description	Comment
200	OK	The request is completed
201	Created	The request has been fulfilled and resulted in a new resource being created
303	See Other	The response should be retrieved from the provided URL
400	Bad request	The server was unable to parse one or more parameters, for example an illegal integer
401	Unauthorized	Basic authentication failed
403	Forbidden	The resource is forbidden for the authenticated client
404	Not Found	The resource does not exist
405	Method not allowed	Unsupported HTTP method used, such as GET when the service requires POST.
406	Not Acceptable	The accept-headers specified requirements that cannot be met.
409	Conflict	The resource is already reserved by a different Client
410	Gone	The resource has been permanently removed. Used for example when the NOD server decides to fail an Order Group permanently.
503	The server is currently unavailable (because it is overloaded or down for maintenance)	This may be returned for any number for system-related reasons such as a system overload. Resubmitting the request MUST respect the Retry-After attribute.
505	HTTP Version Not Supported	Only HTTP-Version/1.1 is supported

Appendix C: NOD Plugins

A plugin is a flexible extension of the NOD server that may serve many purposes that are currently unknown. The requirements use the auxiliary verbs MAY, SHOULD and MUST deliberately to communicate this flexibility within the specification. It is up to the individual plugin provider to decide what of the optional requirements are required in a specific situation.

Scope

This plugin interface specification describes the requirements for the functional interfaces required to provide plugin functionality to the NOD Server. A plugin must however also comply with other non-functional requirements such as performance monitoring, audit logging, quality reviews and performance requirements in order to qualify for deployment in a production setting. These requirements are subject to the plugin request for tender, and are not part of this specification.

This specification does not specify the functionality of the plugins or the contents of the orders they are expected to process (what the plugin should do with a given order), just the common plugin interface that all plugins must adhere to.

Requirements

NOD Plugin definition

- The Plugin is a component deployed on the NOD server that executes a given plugin command. The primary command is execution of an Order in an Order Group by modifying an incoming card image and returning the modified image.
- The Plugin is the only part of PL4 and NOD server that is able to parse the contents of an Order that is submitted to PL4 (in OrderDescription).
- The Plugin is uniquely identified and located by a plugin URI, for example HTTP://127.0.0.1/nod/plugins/hb206/v2
- The Plugin URI is associated with an incoming Order at submission
- The NOD Server provides the plugin with the contents of the card being processed (CARD IMAGE) as well as a context describing attributes of the physical NOD Client (location etc)
- The Plugin REST inferface is stateless; all requests are independent, and the same request should always return the same result. This requirement is not absolute to enable functionality such as fraud detection.
- Plugins can be chained; the output card image of one plugin can be used as input card image for the next.

NOD Plugin Deployment requirements

- A NOD Plugin must expose a REST API at the plugin URI (the URI must therefore be a URL) to execute Plugin Commands. See specificationabove.
- A NOD plugin must expose its own administrative HTTP interface at the {/admin} root.
- The plugin must give access to resources such as the XML schema that will be used by PL4 to validate incoming Orders during mapping before submitting them to the NOD.
- A plugin must explicitly document the widest mapping it supports, so that the PL4-configured Order mappings against the plugin is the same as, or a subset of the supported mappings.
- The web application containing the plugin is a self-contained application;
  - If the plugin requires data from registers, databases etc to execute an Order, these resources should be provided by the Application server independently of the NOD server.
  - When a plugin requires external resources, the plugin developer must also provide the necessary infrastructure such as a database installation. This is is plugin implementation specific, and not covered by this specification.
- In principle any technology may be used to implement the Web Application; Java WAR files, .NET Web Applications or Apache CGI scrips may all be used as long as they implement the REST interfaces. Any requirements restricting this further must be part of the plugin request for tender, and is not part of this specification.

Plugin Configuration

While the plugin command execution is stateless, the Plugins may still have state related to the configuration of the plugin. For example, a plugin may use a local database containing the current zone and fare information in order to execute certain orders. Changes to this configuration will have immediate and retroactive effect on all order groups that already have been submitted to the NOD, and should be avoided unless the consequences have been carefully evaluated by all parties.

- If a plugin has configurable state, the complete configuration of the plugin should be versionable.
- The plugin URI must be used to indicate to the plugin which version of the configuration a plugin command should be executed according to.
  - For example, the URL HTTP://127.0.0.1/nod/plugins/hb206/v2 indicates that the plugin at HTTP://127.0.0.1/nod/plugins/hb206 should execute according to configuration version /v2.

- If the Plugin exposes an administrative interface that allows runtime modification of the configuration, the updated configuration should be made effective only on a new URL eg. at/v3.
  o A plugin may however allow replacing an existing configuration version to correct configuration errors for existing Orders.
- If the Plugin exposes resources that allows runtime modification of the configuration, the updated resources should be made available on a new URL eg. at/v3.

After a versioning of the plugin configuration has occurred, the PL4 mapping should be updated by the PL4 administrator to map future incoming orders to the new URL, thus allowing existing mapped orders to be processed according to the old configuration.

NOD Plugin Security requirements

- A plugin is expected to be deployed in a safe and secure environment. No part of the REST API must be exposed to any party except the NOD Server or administrative functions.
- The plugin ADMIN interface should not be exposed to any party except the Plugin Administrator.
- The plugin ADMIN interface must perform its own HTTP authentication of the Plugin Administrator if it exposes sensitive functionality.
- The plugin may choose to protect resources, and may even allow updating of resources by letting a client POST new versions. The rules for PluginConfiguration must howeverbefollowed.
- If a future plugin is deployed on a server other than the physical NOD servers, a security evaluation must be performed to secure the communication adequately.

NOD PluginRecommendations

- A plugin should be packaged and deployed as an independent web application from other plugins. Multiple operative versions of a given plugin may be bundled in the same web application archive.
- A plugin should have failover support.

Plugin Namespaces

The following namespaces are currently used by NOD Plugins:

Associated Client Namespace	Used by	Description
http://ioas.no/nod/plugin/commands	*	All plugins must support Plugin Commands
http://ioas.no/nod/plugin/commands/processOrder	HB206Plugin	Plugins used for Order execution must support this namespace
http://ioas.no/nod/plugin/transaction	HB206Plugin	Plugins producing DIS transactions must use this namespace
http://ioas.no/nod/plugin/image/desfire	HB206Plugin	Plugins producing or returning DesfireImage datastructures must use this namespace.
http://ioas.no/nod/plugin/commands/getDesfireImageContentsPlugin	DesfireContents	Used by the DesfireContents plugin to describe the contents of an image
http://ioas.no/plugin/commands/message	HB206Plugin	Used by plugins to return localized feedback to the NOD Client

NOD Plugin Commands

All plugins must accept commands from the namespace http://ioas.no/nod/plugin/commands for execution, in a similar fashion to NOD Client Commands. The list of commands may then be specialized with specific namespaces for different plugin functionality.

The commands always have a context element that includes relevant values provided by the NOD Client context.

1-1.1 Example

Plugin ProcessOrder Request with a DesFire Source image

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns5:commands   xmlns:ns2="http://ioas.no/nod/plugin/commands/getDesfireImage-
Contents"
xmlns:ns3="http://ioas.no/nod/plugin/context"
xmlns:ns4="http://ioas.no/nod/plugin/commands/processOrder"
xmlns:ns5="http://ioas.no/nod/plugin/commands"
xmlns:ns6="http://ioas.no/nod/plugin/transaction"
xmlns:ns7="http://ioas.no/nod/plugin/image/desfire">
    <context>
        <ns3:pluginContext>
            <property>
                    <key>capabilityCode</key>
                    <string>11010101</string>
            </property>
                ...
        </ns3:pluginContext>
    </context>
    <command>
        <ns4:processOrder>
            <srcImage>
                    <ns7:image>
                            <application>
                                    <applicationName>TransportDF</applicationName>
                                    ...
                                    <file>
                                            <fileName>T_StoredValue</fileName>
                                            <content>10270000</content>
                                    </file>
                                ...
                            </application>

                        ...
                    </ns7:image>
            </srcImage>
            <orderDescription>PD94bWwgdmVyc....</orderDescription>
        </ns4:processOrder>
    </command>
</ns5:commands>
```

Plugin Response (Target Image and DIS N Transaction returned to NOD Server)

```
<ns4:commands xmlns:ns2="http://ioas.no/nod/plugin/image/desfire"
xmlns:ns3="http://ioas.no/nod/plugin/commands/getDesfireImageContents"
xmlns:ns4="http://ioas.no/nod/plugin/commands"
xmlns:ns5="http://ioas.no/nod/plugin/context"
xmlns:ns6="http://ioas.no/no/plugin/image/desfire/contents"
xmlns:ns7="http://ioas.no/nod/plugin/commands/processOrder"
xmlns:ns8="http://ioas.no/nod/plugin/dis/n"
xmlns:ns9="http://ioas.no/nod/plugin/orderdescription">
  <command>
    <ns7:processOrder>
      <targetImage>
        <ns2:image>
          ...
          <application>
            <applicationName>TransportDF</applicationName>
             ...
            <file>
              <fileName>T_StoredValue</fileName>
              <content>05290000</content>
            </file>
            ...
          </application>
        </ns2:image>
      </targetImage>
      <transaction>
        <ns8:pluginTransaction>
          <ns8:metadata>
            <ns8:disVersion>N</ns8:disVersion>
          </ns8:metadata>
          <ns8:disTransaction>PD94bWwg...</ns8:disTransaction>
        </ns8:pluginTransaction>
      </transaction>
    </ns7:processOrder>
    <result>200</result>
  </command>
</ns4:commands>
```

1-1.2 NOD Plugin Command Result Codes

Each command returns individual results codes. The result codes used by plugin commands follows loosely the number series and error codes in the HTTP specification. This is to improve future compatibility with unknown command sets and capabilities. Note that future plugins may use additional codes, but this must be coordinated with the NOD Server.

The following series are currently supported by the NOD server:

Status Code	Description
1xx	I**nformation from the Plugin/Server.** The contents or results will not trigger any NOD Server state changes.
100	Information from plugin
2xx	OK. This series indicates that the NOD Plugin command was successfully received, understood, and accepted.
200	Execution of the Command was OK.
3xx	Redirection. There are currently no plans to use this series.
4xx	**NOD Server Error.** This series is for errors where the NOD Server seems to have erred. The original command should be included with the command result if possible. The Order Group should not be distributed by the NOD Server anymore.
400	Bad Request, the NOD Plugin could not parse the command. The NOD server should not attempt to resubmit this Plugin Command without modifications. As the NOD Server never should perform an illegal request to the plugin the Order Group must be failed.
401	Command not implemented, the command can not be executed by the NOD Plugin. This may indicate a wrong mapping configuration and the Order Group must be failed.
5xx	NOD Plugin Error. The Plugin is aware that it erred or is incapable of performing the command. The original command should be included with the command result if possible. The Plugin Command may still be re-submitted by the NOD Server.
500	Internal error, an unexpected error occurred on the Plugin.
501	Configuration Error, this may typically be corrected in the Plugin Admin interface so the OrderGroup should not be failed. Note that this error may enable an Order Group to "block" for subsequent orders.

1-2 NOD Plugin REST and ADMIN API

The Plugin document root is the plugin URI, for example HTTP://127.0.0.1/nod/plugins/hb206/v2, the exposed services are relative to this URI.

Method	URL	Payload/Filter	Return Values	Description
POST	/command/execute	Plugin command	plugin command result	The plugin executes and returns the result of a command. Currently only the processOrder command is defined.
GET	/resources/order-Schema.xsd		The schema describing a valid order for this plugin	Plugins processing orders from PL4 must expose this resource. This XML Schema is used by PL4 to validate incoming orders before mapping them against this plugin, and must be available to PL4 without authentication.
*	/resources/*	A resource name	A plugin resource	The plugin may expose resources below this URI to allow RESTful manipulation of these resources. The manipulated resources must respect the Configuration Versioning requirements.
*	/admin			The administrative HTML-based interface for the plugins must be located below this address, and present its welcome page on this URL. This may simply return a redirection.

REST Service /command/execute

Description

Executes the provided commands from namespace http://ioas.no/nod/plugin/commands according to the business rules implemented by the plugin.

The typical case is execution of an Order and return of an updated card image using the command defined in NODPluginProcessOrder.xsd, but other plugin commands may be developed over time.

Note that this service is not strictly a REST service, as is does not actually create any persistent entities as a result of this POST. POST is needed simply because of the size and complexity of the input data structure. The plugin is stateless with respect to this services, and /command/execute simply exposes a calculation as a service.

Note that the context provided in the commands element contains the entries submitted by the client in the client context. The plugin should use these values when relevant, for example using locale and capabilityCode when returning localized messages to NOD.

Special cases

None

URL

POST /command/execute

Supported Request Headers

Headers in addition to the Common REST Interface Specification (App A):

Header	Default	Current Alternatives	Description
Accept	application/xml	none	The returned data structure currently supports XML only.

Parameters

A valid XML-encoded document according to the plugin schema NodPluginCmd.xsd.

Return Values

HTTP Code 200
The XML document with the updated image, transaction and eventual dictionary info according to the plugin_schemain Attachment 1 - NodPluginCmd.xsd.

Response Headers

Headers in addition to the Common REST Interface Specification (App A):

Header	Default	Current Alternatives	Description
Content-Language	NO	none	The Content language in eventual localized feedback from the plugin (dictionary etc)

HTTP Return codes

Codes in addition to the Common REST Interface Specification (App A):

Code	Description	Comment
200	OK	

1-2.1 Example (Content-Type: application/xml):

Plugin Request (Source Image and Order are provided by the NOD Server)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:commands xmlns:ns2="http://ioas.no/nod/plugin/commands"
 xmlns:ns4="http://ioas.no/nod/plugin/dis/j"
xmlns:ns5="http://ioas.no/nod/plugin/image/desfire">
 <command>
  <processOrder>
   <srcImage>
    <ns5:image>
     <application>
      <applicationName>TransportDF</applicationName>
      <file>
       <fileName>T_StoredValue</fileName>
       <content>0000000000000000</content>
      </file>
     </application>
```

```
      </ns5:image>
     </srcImage>
     <orderDescription>PD94bWwgdmVyc2lvbj0iMS4wIiBlbmN....</orderDescription>
    </processOrder>
  </command>
</ns2:commands>
```

Plugin Response (Target Image and Transaction returned to NOD Server)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:commands xmlns:ns2="http://ioas.no/nod/plugin/commands"
 xmlns:ns4="http://ioas.no/nod/plugin/dis/j"
xmlns:ns5="http://ioas.no/nod/plugin/image/desfire">
 <command>
  <processOrder>
   <targetImage>
    <ns5:image>
     <application>
      <applicationName>TransportDF</applicationName>
      <file>
       <fileName>T_StoredValue</fileName>
       <content>03E8</content>
      </file>
     </application>
    </ns5:image>
   </targetImage>
   <transaction>
    <disj:disTransaction xmlns:disj="http://ioas.no/nod/plugin/dis/j">
     <etcEtc>Contents of a DIS J transaction</etcEtc>
    </disj:disTransaction>
   </transaction>
  </processOrder>
  <result>200</result>
 </command>
</ns2:commands>
```

REST Service /resources/*

Description

This service exposes versioned resources made available by the plugin.

The purpose of this interface is to enable easy integration between the plugin and other systems such as PL4, administration, surveillance and automation systems. Different plugins may wish to expose different data to the surrounding systems.

For example: A plugin may expose performance statistics to a surveillance system as resources. The only required resource is currently {/resources/orderSchema.xsd}. This URI must return the XML Schema that should be used by PL4 to validate incoming orders before mapping them against this version of this plugin. All plugins that are capable of processing orders must provide a valid schema description at this URI.

Resources should be cached on the client side according to the Cache-Control header to avoid unnecessary load on the plugin.

Sensitive resources may require authentication.

Special cases

The Plugin may support modification of resources as well. However, the rules for configuration versioning must be followed; the modified resources should not be made available under the current version URI.

For example: The Administrative interface of the plugin may operate by retrieving and manipulating the exposed resource file (e.g {/resources/config.xml}) that represents the plugin configuration state. This will in turn enable easy automatic migration of configuration from TEST to PRODUCTION by an automation system.

Modifying resources is out of scope for the plugin interface specification as it is considered part of the administrative interface of the plugin.

In the rare case that an existing configuration must be modified with immediate retroactive effect (for example due to a configuration error), the plugin may return HTTP 301 to permanently redirect the NOD Server and other clients to a different URI containing a different version number.

URL

GET /resources/<resourceName>

Supported Request Headers

Headers in addition to the Common REST Interface Specification (App A):

Header	Default	Current Alternatives	Description
Accept	*/*	Specified by the individual resource	Some resources may be exposed in several encodings, this is up to the plugin implementation.

Parameters

The Resource name

Return Values

HTTP Code 200
The XML Schema describing a valid order description for this plugin.

Example (Content-Type: application/xml):

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schematargetNamespace="http://io.no/nod/plugins/hb206/v2"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="order">
        <xs:complexType>
            <xs:sequence>
...
```

Response Headers

Headers in addition to the Common REST Interface Specification (App A):

Header	Default	Current Alternatives	Description
Content-Language	NO	none	The Content language in eventual feedback from the plugin (dictionary etc)
Cache-Control	none	max-age=<seconds>, must-revalidate	The period for which the plugin guarantees that this resource will be valid, this should always be considered to avoid unnecessary lookups by external systems. Note however that the effectiveness of HTTP 301 is reduced if resources are cached for a long period of time.

HTTP Return codes

Codes in addition to the Common REST Interface Specification (App A):

Code	Description	Comment
200	OK	
301	MovedPermanently	Used by the plugin to indicate that a different resource configuration should be used. The client must resubmit its request to the specified URL.

Appendix E: NOD Feedback

The NOD supports different kinds of feedback to the NOD Client. This document describes the following feedback mechanisms:

- Messages
- Buzzer
- LED
- Parallel execution

NOD Message Feedback

The NOD Client message command should enable NOD Clients to display context-sensitive localized messages adapted to the NOD Client screen capabilities. In addition to providing text, the message should also provide the NOD Client the possibility of adapting to specific messages, for example by displaying graphics for certain message categories.

To provide high quality text feedback to the NOD Client, several parts of the NOD System must be involved.

- NOD Client command: A command that conveys a message to the Client
- NOD Server: Default messages generated by the NOD
- NOD Plugins: Specific messages generated by the plugins
- PL4 Addorder: Custom override of messages by the calling PTO.

NOD Client Message Command

The NOD Client command message defined in namespace http://ioas.no/nod/client/commands/message enables displaying messages on the NOD Client, and is typically combined with parallel support, for example by flashing LEDs while displaying the message.

In it's basic form, the message will contain the default messages from the NOD message register.

The messages will be adapted to the client capabilities as far as possible using the following input:

- HTTP Header Accept-Language submitted by the NOD Client
- The Capability Codes describing the screen size of the NOD Client.

The following rules apply:

- The message is UTF-8 encoded.
- The client should not support special characters such as newline. Clients supporting textdisplay over several lines will receive one <line> element for each line in the message.

Elements

msgID

The attribute msgID contains a code representing the message (see below)

line

Each line contains a localized message that should be displayed as one continuous line on the NOD Client screen.

Duration

The duration element specifies how long the message should be displayed in milliseconds. The following rules apply:

- The message should be displayed as long as duration specifies (in ms).
- If the card is removed from the NOD Client, the message should be aborted immediately.
- If a min value is provided, the message should be displayed at least this duration regardless of whether the card has been removed or not.

NOD Clients may check for msgID series or specific msgIDs to perform additional customizations on the client side. In the case of specific codes the NOD Client must consult the list of msgIDs a given Plugin or the NOD itself may return.

Example of a NOD Client message command

Display two lines of text on the NOD Client for 4 seconds. If the ecard is removed from the NOD Client while the message is displayed, the message should at least be displayed for 2 seconds.

```
<command cmdID="7">
    <msg:message msgID="hb206.200">
        <line>tPurse kreditert 50kr</line>
        <line>og enkeltreise aktivert</line>
        <duration min="2000">4000</duration>
    </msg:message>
</command>
```

NOD Server

The NOD server provides default messages to the NOD Client. Each message has a msgID, and may be overridden by more specific feedback from the plugins or the Order Group itself (see below). The text content of a msgID is localized and may be changed runtime by the NOD Administrator.

The main job for the NOD Server is to replace its default messages with custom messages if provided, and this mechanism is based on the Message IDs.

The message ID of a message is a 3-digit code for the given message, prefixed by the module from which it originates. A module is typically a plugin, but may also be the NOD system itself. For example, msgIDs that are generated by the HB206 plugin will be prefixed by HB206, and a complete msgID may therefore be HB206.201. The NOD Server uses the prefix NOD.

Both the prefix and the code are decided by the plugin itself, but the combination must be globally unique. This means that different plugins may use the same code for different messages, as long as they use different prefixes.

The following rules apply:

- NOD Establishes the default return message (eg. NOD.200 Order Group executed OK)
- If the plugin returns a message, the NOD default message will be replaced if the code is on the same or a more important series (eg HB206.203: NOK 150 added to tPurse replaces NOD.200 Order Group executed OK).
- If multiple Orders are executed as one Order Group and all message codes are on the same series, NOD will return its default message for this series.
- If the Order Group itself provides custom messages, the current message will be replaced with the custom message, but only if the msgID is identical to the current message. (eg HB206.203: tPurse lower limit reached. 150,- added. replaces HB206.203: NOK 150 added to tPurse)

NOD Message IDs

To enable generic handling of messages by the NOD Clients, the message codes must follow the command result code series of the plugin. The below list includes the current NOD Codes, but also includes example HB206 codes to make the intentions clearer:

Status Code	Description	Module NOD	Module HB206 (examples)
1xx	Information from the Module. The message has not triggered any state changes.		
100	Information to traveller	X	
101	Nothing to do.		X
2xx	OK This series indicates that whatever action the message describes, it was executed OK.		
200	Order Group executed OK	X	
201	Product written		X
202	TPurse modified		X
3xx	Redirection The NOD Server want to direct the Traveller to another resource		
300	See other	X	
4xx	NOD Server Error This series is for errors where the NOD Server is unable to execute the Order Group. The Order Group is removed from dristribution.		
400	Internal Server Error	X	
5xx	NOD Client Error The NOD Server decided that the client is incapable of executing the Order Group. The Order Group may still be distributed by the NOD Server.		
500	Unable to deliver Order Group	X	
501	No space left on media		X

Note that the message text may be modified through the Admin interface of the modules. The text above is just examples. Also note that additional codes may be introduced in the future.

Plugin Message extension

A plugin does not have to support messages. In this case, the NOD default messages will be used. The plugins may provide additional, more detailed messages by using the message command with the http://nod.ioas.no/plugin/commands/message namespace extension.

The following rules apply:

- The message should be localized by the plugin according to the locale context key.
- The number and length of lines must be adapted to the NOD Client capabilities

NOTE: The combined msgID is considered part of the plugins external interface. NOD Clients may check for specific msgIDs to perform additional customizations on the client side, so modules must not change the semantic meaning of a given code unless this is coordinated with the NOD Clients. Each plugin must provide a list of the msgIDs it may produce as part of its specification.

See the Plugin interface description for more details.

Example of a plugin message

```
<processorder>
  <message msgID="HB206.203" >
    <line>tPurse credited NOK 200,-</line>
    <line>Remaining tPurse NOK 350,-</line>
  </message>
...
</processorder>
```

PL4 AddOrder custom message override

While the plugins may give relatively specialized messages, they may not be very user-friendly without having access to registers to convert codes to product names etc. In addition; If an Order Group contains several Orders, a single plugin cannot describe what the entire Order Group execution has achieved as a plugin always executes in the context of a single order.

To facilitate high-quality user feedback the PTO may therefore provide custom messages that overrides the messages given by NOD or its Plugins.

For example, resultcode NOD.200 is default "Order Group Executed"' but the OrderDescription of the Order may override this message with NOD.200: Automatisk påfylt reisepenger NOK 200 i hht avtale.

The following rules apply when the NOD decides what messages to display:

- The locale must match, or the default is chosen
- The msgID must match
- All lines matching the screen capabilities of the NOD Client is chosen
- If no screen capabilities match, the default line is chosen

See the PL4 PTO Client documentation for details of this service.

Example of a PL4 AddOrder message override supporting default and specific screen capabilities as well as an english localization

```
...
 <soapenv:Body>
   <ord1:AddOrdersRequest xmlns:ord1=”http://services.ws.pl4.io.no/orderdomain”>
     <ord1:mediaSerialNumberID>1200006222</ord1:mediaSerialNumberID>

     <message>
      <msgId>HB206.200</msgId>
      <line>
        <value>200kr påfylt reisepenger</value>
      </line>
      <line>
        <screen>011,010</screen>
        <value>Automatisk påfylt reisepenger</value>
      </line>
      <line>
        <screen>011,010</screen>
        <value>NOK 200 i hht avtale</value>
      </line>
      <line>
        <screen>011</screen>
        <value>God tur!</value>
      </line>
     </message>
     <message>
       <locale>en</locale>
       <msgID>HB206.200</msgId>
        <line>
           <value>OK 200 added to tPurse</value>
        </line>
     </message>
      ...
   </ord1:AddOrdersRequest>
 </soapenv:Body>
```

NOD Client LED Support

The NOD Client command led defined in namespace http://ioas.no/nod/client/commands/led enables blinking a coloured led on the NOD Client, and is typically combined with parallel support, for example by flashing LEDs while sounding the buzzer.

Note that this LED command is used by 3 different capabilities, one for each colour. The NOD will only return a led command with a colour supported by the Client Capabilities.

Elements

colour
The colour of the led that should blink

duration
The duration in milliseconds the buzzer should sound

pause
The interval in milliseconds between eventual repetitions. Note that the pause should also be appended after the last repetition.

repeat
The number of times the buzzer should sound

LED Example

Light the red led 1 second, 4 times with a 100ms pause between and after.

```
<command cmdID="5">
    <led:led xmlns:led="http://ioas.no/nod/client/commands/led"
xsi:schemaLocation="http://ioas.no/nod/client/commands/led NODCommandLED.
xsd">
        <colour>red</colour>
        <duration>1000</duration>
        <pause>100</pause>
        <repeat>4</repeat>
    </led:led>
</command>
```

NOD Client Buzzer Support

The NOD Client command buzz defined in namespace http://ioas.no/nod/client/commands/buzzer enables making a buzzing sound on the NOD Client, and is typically combined with parallel support, for example by flashing LEDs while sounding the buzzer.

Elements

freq
The frequency in Hz of the buzzer

duration
The duration in milliseconds the buzzer should sound

pause
The interval in milliseconds between eventual repetitions. Note that the pause also should be appended after the last repetition.

repeat
The number of times the buzzer should sound

Example
Make a buzzing 5kHz sound for one second, repeat 3 times with a 0.1 sec pause between and after.

```
<command cmdID="7">
    <buzz:buzz xmlns:buzz="http://ioas.no/nod/client/commands/buzzer"
xsi:schemaLocation="http://ioas.no/nod/client/commands/buzzer NODCommand-
Buzzer.xsd">
        <freq>5000</freq>
        <duration>1000</duration>
        <pause>100</pause>
        <repeat>3</repeat>
    </buzz:buzz>
</command>
```

Parallel execution of NOD Client Commands on the NOD Client

Some commands may be important to process in parallell to other client commands. For example a light should flash while another operation is performed. To facilitate this, the command parallel from the namespace http://ioas.no/nod/client/commands/parallel may wrap a new set of commands. These commands should be executed in parallel by the NOD Client.

The following rules apply:

- All commands contained in a parallel command set must complete before subsequent commands are processed.
- Parallel commands must always be initiated in the same sequence as they are returned from the NOD Server.
- A parallell command with expectedresult=false should not have to complete before eventual NOD server communication. This enables a NOD Client to communicate with the NOD while eg a LED is blinking.

Resource collisions

If two parallel commands access the same physical resource on the NOD Client, the conflicting commands must be executed in sequence. It is up to the NOD Client to decide when a resource collision has occurred, but the returned result for these commands must be 201 ('Execution of the command was OK, but could not be executed in parallel).

Parallel Execution Example

Light the red and yellow LEDs in parallel, then execute next command.

```
<commands>
    <command cmdID="4" expectedResult="false">
        <parallel:parallel>
            <commands>
                    <command cmdID="5"  expectedResult="false">
                            <led:LED>
                                    <led>red</led>
                                    <duration>1000</duration>
                                    <pause>100</pause>
                                    <repeat>4</repeat>
                            </led:LED>
                    </command>
```

```
                <command cmdID="6"  expectedResult="false">
                        <led:LED>
                                <led>yellow</led>
                                <duration>2000</duration>
                                <pause>100</pause>
                                <repeat>4</repeat>
                        </led:LED>
                </command>
        </commands>
    </parallel:parallel>
  </command>
  <command cmdID="7">
    ...
  </command>
</commands>
```

Appendix F: Static Order Group staticDesfire-Contents

Executing the ”staticDesfireContents” order group enables a NOD client to retrieve a more logical representation of the contents on an ecard.

- When this order group is executed via the POST /group/staticDesfireContents/ nodsession NOD REST interface the NOD server will direct the NOD Client to read a complete card image.
- The NOD Server will then transmit the card image data in a nod image plugin command XML according to the NODPluginImgContentsDesfire.xsd schema from namespace http://ioas.no/no/plugin/image/desfire/contents.
- The NOD image contents plugin will the extract information from the provided card image and generate a logical overview of information stored on the ecard in an XML format according to the NODPluginImgContentsDesfire.xsd schema.
- The NOD Server extracts the logical card contents (base64 encoded) and produces a client response XML according to the NODCommandDesfireContents.xsd schema.

Example NOD Client Request

POST http://ec2-79-125-44-161.eu-west-1.compute.amazonaws.com:8080/nod/client/group/staticDesfireContents/nodsession/ HTTP/1.1
Accept-Encoding: gzip,deflate
Content-Type: application/xml
Authorization: Basic dGVzdDp0ZXN0
Content-Length: 697

```
<?xml version=”1.0” encoding=”UTF-8” standalone=”yes”?>
  <ns6:context xmlns:ns6=”http://ioas.no/nod/client/context”>
</ns6:context>
```

Example NOD Server Response

HTTP/1.1 201 Created
Server: Apache-Coyote/1.1
location: http://ec2-79-125-44-161.eu-west-1.compute.amazonaws.com:8080/nod/client/group/staticDesfireContents/nodsession/cd6aa1ce-bc8a-4f8f-90ec-3571395a8dcb/cmdset/3
Date: Wed, 07 Dec 2011 07:47:02 GMT
Content-Type: application/xml
Transfer-Encoding: chunked

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns3:commands xmlns:ns3="http://ioas.no/nod/client/commands" xmlns:ns9="http://
ioas.no/nod/client/commands/desfirecontents">
    <command cmdID="36" expectedResult="false">
        <ns9:desfireContents>
            <imageDescription>PD94bWwgdmVyc2lvbj0iMS4wIiB....</imageDescription>
        </ns9:desfireContents>
    </command>
</ns3:commands>
```

Example Decoded ImageDescription, as returned in the NodClientCommand desFire-Contents

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns7:imageContents  xmlns:ns3="http://ioas.no/nod/plugin/commands"
xmlns:ns5="http://ioas.no/nod/plugin/commands/getDesfireImageContents"
    xmlns:ns7="http://ioas.no/no/plugin/image/desfire/contents" >
    <contractListFree>7</contractListFree>
    <contract>
        <networkId>5734400</networkId>
        <providerId>3</providerId>
        <tariff>2751</tariff>
        <template>140</template>
        <passengerTotal>1</passengerTotal>
        <autorenew>true</autorenew>
        <priceAmount xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:nil="true" />
        <validityEndDate>2010-04-20+02:00</validityEndDate>
        <originStationId xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:nil="true" />
        <destinationStationId  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:nil="true" />
        <viaStationId xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:nil="true" />
        <status>3</status>
    </contract>
    <expiryDate>2015-12-31+01:00</expiryDate>
    <storedValue>1840</storedValue>
</ns7:imageContents>
```

Statens vegvesen

Håndbøker bestilles fra:

Statens vegvesen Vegdirektoratet
Publikasjonsekspedisjonen
Boks 8142 dep.
0033 Oslo

Telefon: 02030
Faks: 22 07 37 68
publvd@vegvesen.no

ISBN 978-82-7207-639-8