DATELTEK
DANSK TELE TEKNIK

DaTelTek ApS

*i*CH 4 SOAP-API

Release 4.0.1

DaTelTek ApS • Birkevej 4 • DK-4640 Faxe • Denmark • CVR: 31 06 05 59
☎ +45 32 22 22 22 • www.dateltek.dk • ✉ info@dateltek.dk

Table of contents

Change Log 3
*i*CH 4 SOAP API description 4
API calls 4
Authentication 4
Error response 5
Definitions 5
PortingSet 5
Transaction record 5
API Function overview 7
API Lookup functions 9
psetLookup 9
pnoLookup 11
cpsLookup 13
API Porting functions 15
npCreate 15
npConfirm 18
npReject 21
npChange 23
npReturn 26
npCancel 28
Possible error responses from the API: 30
General for all status calls. 31
ooStatus 33
pnoStatus 36
taStatus 37
API Maintenance functions 39
endFlow 39
Transaction states 41
Active states. 41
Passive states. 43

API Error codes and description 45

Simple test client in PHP 47

Change Log

API-Version	Date	Changes
4.0.1	26/10 2011	Fixed problem when counting series in NPCreate, NPChange & NPreturn.
4.0.0	01/10 2011	First official release.
4.0.0 beta 6	24/8 2011	Beta 6 release. Added endFlow facility.
4.0.0 beta 5	23/8 2011	Beta 5 release. Added and corrected Transaction states.
4.0.0 beta 4	22/8 2011	Beta 4 release. Added optional parameter includeAll to pnoStatus.
4.0.0 beta 3	21/8 2011	Third beta release. Changed documentation for pnoStatus.
4.0.0 beta 2	21/7 2011	Second beta release. No further comments.
4.0.0 beta 1	21/7 2011	First beta release.

*i*CH 4 SOAP API description

*i*CH 4 SOAP API (hereafter API) is used for communicating with DaTelTek ApS *i*CH 4-manager from external applications through calls to the API engine using general SOAP protocol.

Using the API enables possibility to achieve the same functionality in the ICH system as if using the WEB based interface.

API calls

All functionality of the API is documented in the WSDL that can be optained in the following manner:

http://<ICH_URL>:8080?wsdl

Furthermore all public API functions is revealed by accesing the following URL:

http://<ICH_URL>:8080

This documentation will only focus on the parsing of parameters to and from the API and the dependencies in between these.

In general, al parameters, names, values and procedures is described in the public accessable documents

Rules & Procedures on Number Portability, V2.2, May 2011
(http://www.teleindu.dk/billeder/Nummerportabilitet/RulesProcedure_-__APG26_022-final.pdf)

Requirements/Transactions for Number Portability, V1.9, March 2008
(http://www.teleindu.dk/billeder/Nummerportabilitet/APG96_019-final20080327.pdf)

All **valid** calls to the API will return either the requested dataset or an error message. Any **non-valid** calls will return an SOAP exception error.

Examples in this documentation is written in PHP language.

Authentication

The API us standard HTTP authentication and is used in any call except fetching of the WSDL.

Authentication example:

```
<?php

      $ConnectOptions = array (
                    'login' => 'api_user',
                    'password' => 'api_pass');

      $WSDL = "http://<ICH_URL>:8080?wsdl";
      $apiClient = new SoapClient($WSDL, $ConnectOptions);

?>
```

Error response

An error from the API will have the following format:

```
stdClass Object
(
    [errorCode] => errNo
    [errorText] => errText
)
```

The API will always return one, and only one errorClass

Possible errorcodes is found later in this document.

Definitions

In general all API calls are not case sensitive.

Besides the definitions found in “Rules and Procedures” and “Requirements for transactions” in relation to OCH Denmark, the following terms is used in this documentation:

PortingSet

A dataset containing RoutingInfo, ChargingInfo, SPC, Municipality, Serviceoperator and NetworkOperator. Several “Portingsets” may be created in the system allowing NetworkOperators to interconnect with several carriers and to have several ServiceOperators attached with them as NetworkOperators. Also this allows for several SPC nodes (on SS7 level) and or mobile carriers to which the numbers should be ported.

NetworkOperators could use the API – through own systems – to service porting requests for ServiceOperators, and could take great advantage of creating a PortingSet for each of these operators.

PortingSets are created and maintained by DaTelTek technical staff. There is no support for creating or modifying these neither through the API nor the Web-based interface as this requires further licensing adding service operators to the system.

Transaction record

All transactions to and from OCH is described in a transaction. This transaction contains all needed information send or received and a current “state” that described the immediate status for the transactions progress in the system.

A typical transaction flow could be:

1. We send a npCreate to OCH to request the start of a porting. At this time the only reference to the transaction (besides the telephone number requested) will be our OriginatingOrderNumber, a unique number we are required to assign all outgoing transactions towards OCH. The API will respond this number upon start of a porting request.
2. OCH will validate the transaction and either replies to the ICH system with an npError in case of malfunction, faulty parameters passed or conflict with another open flow, or if successfully parsed

– an OCH_OrderResponse confirming the reception of the transaction and the start of a flow in OCH. In the OCH_OrderResponse the OCHOrderNumber is found, this is the OCH transaction number that may be used for reference in the entire lifetime of any transaction in OCH related to the initial request and may be used to follow the flow in OCH-online. Typically the OCH_OrderResponse (or npError in case of problems) will be responded within 3-5 minutes after initiating the request.

3. Now the ICH transaction will enter a state where we await either a npConfirm or npReject from the donor of the number. This will typically take 5-15 days (as described in "Rules and Procedures"). If the porting is rejected by the donor, the ICH system will receive a npReject with a cause for the rejection and the entire flow is terminated on OCH. The user of the ICH system is informed and will have to take appropriate actions both in customer relation and in the ICH by closing the flow locally. In case the customer should regret the porting or an error was made during the initial request, a npCancel may be issued by us which will cancel the entire flow on both the Donor's ICH system, our ICH system and OCH.

1. When the agreed date and time for porting the number is reached, the ICH system will update own database and automatically submit an npComplete to OCH. This will be translated by OCH to an npUpdate which is then send to all active operators for them to update their own ICH database with the new location of the ported number(s).

2. All other Service- and Network operators will reply with a npUpdateComplete to OCH – a reply that is relayed to us as recipient of the number, to inform us that their ICH system has been updated correctly with the information related to the number porting issued. This will normally take 15-20 minutes after we issued the npComplete, depending on the load of the other operators ICH systems. During this time we may request a list in our ICH system describing which operators that have not yet responded.

3. The number is now ported (when all other operators have responded) and all flows in both ICH and OCH is terminated normally. In case an operator fails to respond, OCH will issue a pseudo npUpdateComplete on their behalf after a number of days (typically 7 days). This to ensure that the flow is terminated normally within a reasonable timeframe.

API Function overview

Function call	Reply from API	Comments	Category
about	API will reply with current version.	Good for testing.	General
apiservertime	Return the current timestamp of the ICH server.	Good for testing.	General
releaseInfo	Return the audit for the API.	Informal Good for testing.	General
psetLookup	SPC, Municipality, RoutingInfo, ChargingInfo, Serviceoperator and Netværksoperator for each PortingSet. Error if no PortingSet is found.	Lookup all active PortingSet's in the database.	General Lookup
pnoLookup	All database information related to the number (serie). Especially Serviceoperator, Netværksoperator, PortingCase, NumberType, SPC, Municipality, RoutingInfo and ChargingInfo may be useable upon porting request.	Lookup details related to a telephone number	General Lookup
cpsLookup	The name and possible other information about an Operator is replied.	Lookup the information related to an operator based on his CPS code.	General Lookup
npCreate	Our OriginatingOrderNumber for the transaction or an error.	Starts a port of a number (series)	Porting
npChange	Our OriginatingOrderNumber for the transaction or an error.	Start changing information on a phone number. (May be used instead of npCreate while porting between own passive serviceoperators (Using PortingSet's for own LSO's)	Update / Porting
npReturn	Our OriginatingOrderNumber for the transaction or an error.	Return a number to range holder (after 6 months retention) when a customer relation is terminated.	Update

npCancel	Our OriginatingOrderNumber for the transaction or an error.	Cancel an active flow if customer regrets or if an error was made.	Update
ooStatus	One or more transaction records	Show status based on OriginatingOrdernumber	Status
pnoStatus	One or more transaction records	Show status based on telephone number	Status
taStatus	One or more transaction records	Show list of active transactions based on transaction state	Status
endFlow	Our OriginatingOrderNumber for the transaction or an error.	Local termination (ICH) on a porting-flow	Maintenance

API Lookup functions

psetLookup

Function: **Lookup PortingSet**

Description: **Fetch values from the PortingSet table**

Function call	Mandatory values	Optional values	Comments
psetLookup			

Response value(s)	Comments
	A 'PortingSet' record for each PortingSet in the ICH database is returned
name	The name on the PortingSet
spc	SPC "Signalling Point Code" (SS7 node number) – The number of the interconnect node to which a customer is attached. Must be used in conjunction with municipality and voids the use of Routing- / Charging information
municipality	Municipality – legacy information related to the geographic location of the customer. Must be used in conjunction with SPC and voids the use of Routing- / Charging information
routingInfo	RoutingInfo – mostly used for Mobile phone numbers and must be used in conjunction with ChargingInfo. Voids the use of SPC/Municipality.
chargingInfo	ChargingInfo – mostly used for Mobile phone numbers and must be used in conjunction with RoutingInfo. Voids the use of SPC/Municipality.
serviceOperator	The Serviceoperator that will be the owner of the number
networkOperator	The Networksoperator that should route the calls for the number

Example on usage of psetLookup:

Response from the API:

```
stdClass Object
(
    [count] => 2
    [portingSets] => Array
        (
            [0] => stdClass Object
                (
                    [name] => TeleCom Fixed
                    [spc] => 00
                    [municipality] => 000
                    [routingInfo] => 7273
                    [chargingInfo] => 7273
                    [networkOperator] => 01072
                    [serviceOperator] => 01072
                )

            [1] => stdClass Object
                (
                    [name] => TeleCom FIXED 2
                    [spc] => 00
                    [municipality] => 000
                    [routingInfo] => 703450
                    [chargingInfo] => 703450
                    [networkOperator] => 01072
                    [serviceOperator] => 01072
                )

        )

)
```

Possible error responses from the API:

Errorcode	Comments
30	”No Portingsets found in DataBase”

pnoLookup

Function: **Lookup PhoneNumber**

Description: **Fetch values for a telephonenumber in the database**

Function call	Mandatory values	Optional values	Comments
pnoLookup			
	Phonenumber		The telephone number for information request

Response value(s)	Comments
	All information found in database related to the given number is responded
entryType	Either ”R” or ”P” for respective Range or Ported entry in Database
rangeHolder	The operator that subscribe to the Range of numbers from the regulatory agency to which the number(s) originates
serviceOperator	The Serviceoperator that is registered as having the customer relationship related to the number(s)
networkOperator	The Networksoperator that routes the calls for the number(s)
firstPhoneNumber	The first number in the series (Does NOT necessarily mean that the number is part of a series – may be same as LastPhoneNumber)
lastPhoneNumber	The first number in the series (Does NOT necessarily mean that the number is part of a series – may be same as FirstPhoneNumber)
portingCase	May contain the values ’PortedNonGeo’, ’PortedWithGeo’ or ’NonPorted’ as described in “Requirements for transactions” by OCH
municipality	Municipality (if the operator use RoutingInfo/ ChargingInfo this value will always be ’000’)
spc	SPC (if the operator use RoutingInfo/ ChargingInfo this value will always be ’00’)
numberType	May contain the values ’FIXED’ or ’GSM’ there is no special indication for VoIP.
routingInfo	RoutingInfo (if the operator use SPC / Municipality this value will always be ’00000000’)
chargingInfo	ChargingInfo (if the operator use SPC / Municipality this value will always be ’00000000’)
lubo	Internal database value – Last updating operator

Example on usage of pnoLookup:

Response from the API:

```
stdClass Object
(
    [entryType] => P
    [rangeHolder] => 01011
    [networkOperator] => 01083
    [serviceOperator] => 01083
    [firstPhoneNumber] => 56574016
    [lastPhoneNumber] => 56574016
    [portingCase] => PortedWithGeo
    [municipality] => 740
    [spc] => 29601
    [numberType] => FIXED
    [routingInfo] => 00000000
    [chargingInfo] => 00000000
    [lubo] => 01083
)
```

Possible error responses from the API:

Errorcode	Comments
1	”Telephone number is not numeric”
2	”Telephone number is not 8 ciphers”
3	” Telephone number is invalid”

cpsLookup

Function: **Operator Lookup**

Description: **Fetch operator information based on CPS code**

Function call	Mandatory values	Optional values	Comments
cpsLookup			
	CPS		The operators unique CPS code as assigned by OCH

Response value(s)	Comments
desc	The name for the Operator as registered in ICH (data typically fetched from OCH)
active	0/1 – Describes wether the operator is active on OCH
contactPerson	If registered – the name of the NP responsible at the operator
email	If registered –E-mail of the NP responsible at the operator
phoneNumber	If registered – If registered, the contact telephonenumber of the NP responsible at the operator
mobileNumber	If registered – Cell phone number of the NP responsible at the operator
faxNumber	If registered –FAX number of the NP responsible at the operator
opeType	Type of Operator

Example on usage of cpsLookup:

Response from the API:

```
stdClass Object
(
    [desc] => TDC
    [active] => 1
    [contactPerson] => Carsten Hviid
    [email] => cah@tdc.dk
    [phoneNumber] => 66641462
    [mobileNumber] => 21462123
    [faxNumber] =>
    [opeType] => 1
)
```

Note that no value for <FaxNumber> is replied in this example

Possible error responses from the API:

Errorcode	Comments
6	”Not found among serviceoperators”

API Porting functions

npCreate

Function: **Create (initiate) NumberPorting**

Description: **Initiate the portering of a number as recipient**

The only mandatory parameter is PhoneNumber. If no other parameters is passed, Numberporting is initiated of this (and only this) number with preset default values. If PortingSet is entered, SPC, Municipality, RoutingInfo, ChargingInfo, RecipientServiceOperator and RecipientNetworkOperator will be fetched from the PortingSet entered. All values entered in the XML parameters will override default information fetched if entered.

Function call	Mandatory values	Optional values	Comments
npCreate			
	phoneNumber		The number that is requested (may be primary number in Type II porting of series)
		portingSet	PorteringSet requested for this porting
		routingInfo	
		chargingInfo	
		spc	
		municipality	
		recipientNetworkOperator	
		recipientServiceOperator	
		currentServiceOperator	
		currentNumberType	
		pointOfConnection	
		customerID	Only in case of PrePaid
		icc	Only in case of PrePaid
		requestedExecutionDate	Format: **YYYYMMDD**

		requestedExecutionTime	format: **HHMM**
		seriesCount	Number of series if Type II
		series 0-n	If Type II, use the format: **hgfedcba-hgfedcba** **hgfedcba-hgfedcba** **.** **.** **hgfedcba-hgfedcba**

Response value(s)	**Comments**
originatingOrdernumber	Our ordernumber on the transaction assigned by the ICH system

Example on usage of npCreate:

Response from the API:

```
stdClass Object
(
    [originatingOrderNumber] => 0108361046
)
```

Possible error responses from the API:

Errorcode	Comments
1	”Telephone number is not numeric”
2	”Telephone number is not 8 ciphers”
3	”Telephone number is invalid”
4	”The second telephone number is less than the first telephone number in series
7	”Not registered as a network operator”
8	”Was not found among network operators”
9	”Not a valid date format (YYYYMMDD)”
10	”Date is not numeric (YYYYMMDD)”
11	”Date is in the past”
12	”Error in time format (HHMM)”
13	”SeriesCount is wrongly calculated”
14	”We are already operator of the number”
15	”Number is present in another open flow”
17	”Series format incorrect (hgfedcba- hgfedcba)”
32	”The given PortingSet is not found in database”
40	”Series missing (SeriesCount and number of given series mismatch)”

npConfirm

Function: **Confirm NumberPorting**

Description: **Accept the porting of a number as donor**

Used as response on an incomming npCreate where we are donor. Only mandatory parameter is EITHER PhoneNumber OR OCHOrderNumber OR OrigOrdernumber as long as the – by recipient requested – date for execution is not changed. If nothing else is passed as parameters, this (and only this) outgoing port is accepted with EITHER the requested date (from recipient) OR "tomorrow" in case no date for execution was requested.

IF the recipient has requested a date for execution and we as donor wish to alter this date (for any reason) the parameter" Confirmed Status" becomes mandatory.

NOTE that the rules for postponing execution of requested date from recipient MUST follow OCH's rules for this and that npConfirm in the opposite case WILL be rejected by the API.

Function call	Mandatory values	Optional values	Comments
npConfirm			
	phoneNumber OR ochOrderNumber OR origOrderNumber		The value the transaction is identified by. Precedens in case of several entries is: 1. PhoneNumber 2. OriginatingOrderNumber 3. OCHOrderNumber
	directoryStatus		Indicates how the number was listed with us (directory) Values Unlisted, Secure and so on – IF number is a special secure listed number AND fixed, this parameter becomes mandatory
		confirmedExecutionDate	The date we accept for execution of the port. Note that if this date is different than the date requested, ConfirmedStatus become mandatory
	confirmationStatus		See above.

Response value(s)	Comments
OCHOrdernumber	OCH's ordernumber related to this flow

Example on usage of npConfirm:

Response from the API:

```
stdClass Object
(
    [ochOrderNumber] => 2001081046
)
```

Possible error responses from the API:

Errorcode	Comments
80	”Either telephone number, OCHOrderNumber or OriginatingOrderNumber must be passed”
82	” Parent transaction not found”
83	”In re-confirm, the confirmed date must be later than the requested.”
84	”ConfirmationStatus must be entered if confirmed date is different from requested”
86	”ConfirmationStatus must be a value between 1 and 4” – see “Requirements for transactions” by OCH for details regarding this parameter
87	”ConfirmedExecutionDate is in the past”
88	” ConfirmedExecutionDate is more than 30 days in the future from requested”
89	”ConfirmedExecutionDate may not be earlier than RequestedExecutionDate”

npReject

Function: **Reject NumberPorting**

Description: **Reject a porting of a number as donor**

Used as response on an incoming npCreate where we are donor. Only mandatory parameter is EITHER PhoneNumber OR OCHOrderNumber OR OrigOrdernumber.

A reject code must be passed to recipient reasoning the reject. See “Requirements for transactions” and “Rules and procedures” by OCH regarding details for causing rejects.

Function call	Mandatory values	Optional values	Comments
npReject			
	phoneNumber OR ochOrderNumber OR origOrderNumber		The value used for looking up the transaction. Precedence is: 1. PhoneNumber 2. OriginatingOrderNumber 3. OCHOrderNumber
	rejectCode		See the section ”Reject codes” for these values and OCH documentation for valid causes for rejects

Response value(s)	Comments
OCHOrdernumber	OCH’s ordernumber related to this flow

Example on usage of npReject:

Response from the API:

```
stdClass Object
(
    [ochOrderNumber] => 2001081346
)
```

Possible error responses from the API:

Errorcode	Comments
80	”Either telephone number, OCHOrderNumber or OriginatingOrderNumber must be passed”
82	” Parent transaction not found”
92	”Reject code missing”
94	”Reject code not found among valid codes”

npChange

Function: **Change number**

Description: **Change a number or series of numbers in the OCH (and ICH) Numberdatabase**

Only mandatory parameter is PhoneNumber, but if no other parameter is passed, no changes will be made. First all informations related to the number(s) are fetched and then these are then overwritten with all optional parameters passed in this current XML transaction. A network operator may change CurrentServiceOperator, SPC, NumberType, Municipality, ChargingInfo, RoutingInfo and NumberPorted indicator. A serviceoperator may change CurrentServiceOperator and the NumberPorted indicator.

This function is used to ”port” a telephone number between passive ServiceOperators (LSO’s) under the same NetworkOperator instead of npCreate. The API MUST be used this way to ensure correct communication with OCH while NetworkOperator is same and only ServiceOperator is changed.

Function call	Mandatory values	Optional values	Comments
npChange			
	phonenumber		The number the changes should concern.
		currentServiceOperator	
		currentNetworkOperator	
		recipientNetworkOperator	
		portingCase	
		spc	
		municipality	
		routingInfo	
		chargingInfo	
		newNumberType	
		numberPorted	
		seriesCount	Number of series if Type II

		series_n	If Type II, use the format: **hgfedcba-hgfedcba** **hgfedcba-hgfedcba** **.** **.** **hgfedcba-hgfedcba**

Response value(s)	**Comments**
OriginatingOrdernumber	Our ordernumber for the transaction.

Example on usage of npChange:

Response from the API:

```
stdClass Object
(
    [originatingOrderNumber] => 0103361046
)
```

Possible error responses from the API:

Errorcode	Comments
1	”Telephone number is not numeric”
2	”Telephone number is not 8 ciphers”
3	”Telephone number is invalid”
4	”The second telephone number is less than the first telephone number in series
7	”Not registered as a network operator”
8	”Was not found among network operators”
13	”SeriesCount is wrongly calculated”
15	”Number is present in another open flow”
17	”Series format incorrect (hgfedcba- hgfedcba)”
18	”We are not operator on the number”
40	”Series missing (SeriesCount and number of given series mismatch)”

npReturn

Function: **Return number**

Description: **Retur a telephone number to RangeHolder upon Customer termination.**

Only mandatory parameter is PhoneNumber, if series are attached to the number (Type II) these must be passed.

DaTelTek *i*CH-manager supports ”delayed return of telephone number” so the retention time is held after customer termination. To uset his feature, the parameter ”ReturnDate” with the required ”RunDate” for the return of number must be passed. (today this is 6 months atfer termination).

The return of number series (Type II) follow the normal indications for these.

Function call	Mandatory values	Optional values	Comments
npReturn			
	phoneNumber		The number that must be returned.
		returnDate	The date the system should send the transaction to OCH. If not applied, the number will be returned immidiately. Format: **YYYYMMDD**
		seriesCount	Number of series if Type II
		series_n	If the number is Type II the following format is used: **hgfedcba-hgfedcba** **hgfedcba-hgfedcba** **.** **.** **hgfedcba-hgfedcba**

Response value(s)	Comments
OriginatingOrdernumber	Our ordernumber for the transaction.

Example on usage of npReturn:

Response from the API:

```
stdClass Object
(
    [originatingOrderNumber] => 0103361046
)
```

npCancel

Function: **Cancel NumberPorting**

Description: **Cancel an already started porting of telephone number as recipient**

Mandatory is either OriginatingOrdernumber or PhoneNumber.

npCancel may be used if our customer regrets or there is something wrong with the creation of the port of telephone number(s). May ONLY be used on a transaction that has not surpassed an accepted date of port by donor (PONS).

Function call	Mandatory values	Optional values	Comments
npcancel			
	originatingorderNumber **or**		Our originale ordrenumber for the port.
	phoneNumber		Phone number. If BOTH phone number and OriginatingOrdernumber is applied, only the phone number will be used.

Response value(s)	Comments
OriginatingOrdernumber	Our ordernumber for the transaction.

Example on usage of npCancel:

Response from the API:

```
stdClass Object
(
    [originatingOrderNumber] => 0103361046
)
```

Possible error responses from the API:

Errorcode	Comments
1	”Telephone number is not numeric”
2	”Telephone number is not 8 ciphers”
3	”Telephone number is invalid”
4	”The second telephone number is less than the first telephone number in series
50	”originating ordernumber was not found as an active transaction”
60	”Either telephone number or originatingordernumber must be passed”
62	”Telephone number not found among active transactions”
64	”More than one transaction found, must be handled manually.”
66	”transaction is not in a state that allow npCancel”

Possible error responses from the API:

Errorcode	Comments
1	”Telephone number is not numeric”
2	”Telephone number is not 8 ciphers”
3	”Telephone number is invalid”
4	”The second telephone number is less than the first telephone number in series
13	”SeriesCount is wrongly calculated”
15	”Number is present in another open flow”
17	”Series format incorrect (hgfedcba- hgfedcba)”
18	”We are not operator on the number”
40	”Series missing (SeriesCount and number of given series mismatch)”

API Status functions

General for all status calls.

All status calls return a record from the transaction database that has the following general layout:

Response value(s)	Comments
Id	The ID for the transaction from the database.
state	All transactions is in a state that describes the current progress. For details see the section ”Transaction states”
direction	Indicates whether the actual transaction is an incoming from OCG (=’I’) or outgoing towards OCH (=’O’)
transactiontype	The Transactionstype. 001 npCreate 002 npOCH_Order_Response 004 npConfirmation 005 npError 006 npReject 007 npCancel 008 npComplete 009 npUpdate 010 npUpdateComplete 012 npReturn 014 npRangeUpdate 017 npChange 018 npPortingRequest **(IMPLEMENTATION NOT TESTED)** 019 npPortingResponse **(IMPLEMENTATION NOT TESTED)** Please read the document ”Requirement for transactions” and ”Rules and Procedures” issued by OCH for details.
date	Date for initiating the transaction.
time	Time for initiating the transaction
priority	Transaction priority in OCH (2 or 5)
ochOrderNumber	Ordrenumber assigned by OCH. Will not be issued before OCH_Order_Response is received from OCH.
origordernumber	The unique internal order number we assign any transaction initiated towards OCH.
uniqueID	The Transaction unique ID – assigned by OCH on every accepted transaction.
telePhoneNumber	Telephone number or primary number in Type II series related to this transaction.
message	The complete message (transaction) that was sent to – or received from OCH.

rundate	The date for the execution of the transaction. (0 = immidiately)
internal	Internal field containing information for later dispatch of npUpdateComplete

ooStatus

Function: **Lookup OriginatingOrdernumber**

description: **Lookup status for an transaction based on OriginatingOrdernumber (our unique ID)**

This function may be used to lookup a transaction or list of transactions based OriginatingOrderNumber. The function will reply a number of transactions specified in ”records_in_list” or an error if none found

Function call	Mandatory values	Optional values	Comments
ooStatus			
	originatingOrdernumber		Eg. Returned by npCreate, npChange or similar.
		recordOffset	Offset in recordlist when returning > 100 elements in list

Response value(s)	Comments
	While state = 0, the transaction has not been processed by the ICH-system. **For further information please see the section: ”General for all status calls”**
recordsInList	Number of records returned by API
totalRecords	Total number of records in dataquery
currentOffset	Requested offset into recordList
records	Returned records

Example on usage of ooStatus

Response from the API:

```
stdClass Object
(
    [recordsInList] => 2
    [totalRecords] => 2
    [currentOffset] => 0
    [records] => Array
        (
            [0] => stdClass Object
                (
                    [id] => 6099528
                    [state] => 2111
                    [direction] => I
                    [transactionType] => 001
                    [date] => 2011-07-05
                    [time] => 14:16:00
                    [priority] => 5
                    [ochOrderNumber] => 200015714897
                    [origOrderNumber] => 01079000000000035278
                    [uniqueID] => 880451455
                    [telePhoneNumber] => 44351314
                    [message] => TransactionType=001;
TelephoneNumber=44351314;
OCHOrderNumber=200015714897;
UniqueID=880451455;
OriginatingOrderNumber=01079000000000035278;
CurrentServiceOperator=01096;
RecipientServiceOperator=01079;
RecipientNetworkOperator=01079;
CurrentNumberType=FIXED;
RequestedExecutionDate=20110711;
RequestedExecutionTime=0800;
CustomerID=25250;
PointOfConnection=RECIPIENT;
SeriesCount=0;

                    [rundate] => 2000-00-00 00:00:00
                    [internal] =>
                )

            [1] => stdClass Object
                (
                    [id] => 6259180
                    [state] => 2111
                    [direction] => I
                    [transactionType] => 001
                    [date] => 2011-07-15
                    [time] => 15:08:00
                    [priority] => 5
                    [ochOrderNumber] => 200015781833
                    [origOrderNumber] => 01010000401770
                    [uniqueID] => 891315590
                    [telePhoneNumber] => 44351314
                    [message] => TransactionType=001;
TelephoneNumber=44351314;
OCHOrderNumber=200015781833;
UniqueID=891315590;
OriginatingOrderNumber=01010000401770;
CurrentServiceOperator=01096;
RecipientServiceOperator=08077;
RecipientNetworkOperator=01010;
CurrentNumberType=FIXED;
PointOfConnection=RECIPIENT;
SeriesCount=0;

                    [rundate] => 2000-00-00 00:00:00
                    [internal] =>
                )
        )
)
```

Possible error responses from the API:

Errorcode	Comments
50	”Originating Ordernumber was not found as an active transaction”
52	” Originating Ordernumber was not found in the list of parameters”

pnoStatus

Function: **Lookup PhoneNumber status**

Description: **Find current status based on a telephone number**

This function may be used to lookup a transaction (or list of transactions) based on a telephonenumber. The function will reply a number of transactions specified in ”records_in_list” or recordsInList = totalRecords = currentOffset = 0 if none records found.

Function call	Mandatory values	Optional values	Comments
pnoStatus			
	phoneNumber		
		recordOffset	Offset in recordlist when returning > 100 elements in list
		includeAll	If 1, return all records, including processStatus = DONE (=9999)

Response value(s)	Comments
recordsInList	Number of records returned by API
totalRecords	Total number of records in dataquery
currentOffset	Requested offset into recordList
records	Returned records
	For further information please see the section: ”General for all status calls” and the parameter set described in OO_Status

Same parameter set as for ooStatus + ConfirmedExecutionDate possible ConfirmedExecutionTime (if any) if the transaction is an npCreate with us as Donor, and npConfirm(s) has been replied. E.g.

Possible error responses from the API:

Errorcode	Comments
54	”Telephonenumber was not found in the list of parameters”

taStatus

Function: **Lookup Transaction status**

Description: **Return status from the transaction database.**

This function may be used to fetch a list of ongoing transactions all matching the criteria specified. There may only be entered one of the available parameters per request.

Function call	Mandatory values	Optional values	Comments
taStatus			
	taType (one of the following): **AllOpen** **WaitOCHresponse** **WaitDonorResponse** **WaitExecution** **WaitOtherOpe** **WaitOurResponse** **Cancelled** **Rejected** **Errors**		Matching states: All != 9999 21, 310, 410, (510) 1022 1023, 1024, (1623) 1220, 1320, 1420, (1510) 110, [1023], [1024], [2111] 9127 8550 8026, 8027, 8226, 8326, 8426, 8526, 8550, 9026 NOTE: [xxxx] specifies the possibility to resend npConfirm with a new date. (xxxx) concern npPortReq and has not yet been tested in implementation.

Response value(s)	Comments
recordsInList	Number of records returned by API
totalRecords	Total number of records in dataquery
currentOffset	Requested offset into recordList
records	Returned records
	For further information please see the section: ”General for all status calls” and the parameter set described in OO_Status **NOTE: WaitOurResponse responds both new incomming npCreates awaiting npConfirm or npReject and npCreates to which an npConfirm already has been send. These may be identified by checking transaction->state. This will be 110 for new – not already responded npCreates – and 2111 for npCreates to which an npConfirm already has been send.**

Same parameter set as for ooStatus

Possible error responses from the API:

Errorcode	Comments
56	”TA_type was not found in the list of parameters”
58	”Invalid TA_type”
70	”No transactions found in the state requested”

API Maintenance functions

endFlow

Function: **End flow locally in iCH system**

Description: **End a internal flow in iCH system**

The only mandatory parameter is OriginatingOrdernumer or OCHorderNumber. It is only possible to end flows that has already is ended at OCH, e.g. flows that has been rejected by donor on only wait for a iCH accept.

Function call	Mandatory values	Optional values	Comments
endFlowCreate			
	originatingorderNumber		The originatingOrderNumber we assigned the flow
	ochorderNumber		The ochorderNumber assigned by OCH to flow.

Response value(s)	Comments
originatingOrdernumber	Our ordernumber on the transaction assigned by the ICH system

Example on usage of endFlowCreate:

Response from the API:

```
stdClass Object
(
    [originatingOrderNumber] => 0108361046
)
```

Possible error responses from the API:

Errorcode	Comments
75	” Either OCHOrderNumber or OriginatingOrderNumber must be passed to endFlow”
77	” Only OCHOrderNumber or OriginatingOrderNumber, not both, must be passed to endFlow”
78	” Transaction not in a state where endFlow allowed (state must be less than 8000)”

Transaction states

Active states.

The value ”state” indicates the current state of the transaction at hand and may have the following values:

Value	State	Category	Description
0	NON_PROC		A not processed transaction.
21	WAIT_NP_OCH_RESP	npCreate **We are recipient**	npCreate send awaiting OCH_Order_Response
110	WAIT_NP_CREAT_GUI	npCreate **We are Donor**	npCreate received from OCH. Awaiting accept/reject
210	WAIT_RU_OCH_RESP	npRangeUpd	npRangeUpd send to OCH – awaiting OCH_Order_Response
310	WAIT_RET_OCH_RESP	npReturn	npReturn send to OCH, awaiting OCH_Order_Response
410	WAIT_NPC_OCH_RESP	npChange	npChange send to OCH, awaiting OCH_Order_Response
510	WAIT_NPR_OCH_RESP	npPortRequest **We are recipient**	npPortRequest send to OCH, awaiting OCH_Order_Response **(NOT TESTED IN IMPLEMENTATION)**
710	WAIT_NP_CONF_ERR	npConfirm **We are recipient**	npError recieved to a formely send NP_Confirm, Flow still open
Point Of No Stop (PONS)			
1022	WAIT_NP_CONF_REJ	npCreate **We are recipient**	Awaiting npConfirm or npReject from Donor
1023	WAIT_NP_EXEC_DATE	npCreate **We are recipient**	Awaiting date for porting
1024	WAIT_NP_EXEC_DATE2	npCreate **We are recipient**	Awaiting date for porting if ConfirmedExecutionDate is different than RequestedDate (changed by donor).
1220	WAIT_FOR_ALL_RUPD	npRangeUpd	Awaiting npUpdateComplete from all other operators upon initialization of numberserie.
1320	WAIT_FOR_ALL_RETUPD	npRangeUpd	Awaiting npUpdateComplete from all other operators
1420	WAIT_FOR_ALL_CHGUPD	npRangeUpd	Awaiting npUpdateComplete from all other operateors related to changes in number series.

1510	WAIT_NP_PORT_RESP	npPortRequest **We are Recipient**	Awaiting npPortingResponse from Network operator **(Not tested in implementation)**
1540	WAIT_NP_PORT_START	npPortRequest **We are Recipient**	Awaiting Network operator to start npCreate flow **(Not tested in implementation)**
1622	CC_WAIT_NP_CONF_REJ	npPortRequest **We are Recipient**	Awaiting cc:npConfirm or cc:npReject from Donor **(Not tested in implementation)**
1623	CC_WAIT_NP_UPD	npPortRequest **We are Recipient**	Awaiting execution date on porting started by npPortRequest **(Not tested in implementation)**
Point Of No Return (PONR)			
2024	WAIT_NP_UPD_COMPL	npCreate	Date for Porting reached, npUpdate send to OCH, awaiting npUpdateComplete from all other operators
2111	WAIT_NP_UPD_REC	npCreate **We are Donor**	npConfirm send. Await porteringsdate (or send new npConfirm)
8026	WAIT_NP_ERR_GUICL		npError received from OCH. Awaiting reaction from staff.
8027	WAIT_NP_CPL_GUICL		npError received from OCH. Received from OCH as response to npComplete. Awaiting reaction from staff. (Typically upon errors in Routing/Charging Info, SPC/Municipality or PortingCase)
8226	WAIT_RU_ERR_GUICL	npRangeUpd	npError received. Awaiting reaction from staff.
8326	WAIT_RET_ERR_GUICL	npReturn	npError received. Awaiting reaction from staff.
8426	WAIT_CHG_ERR_GUICL	npChange	npError received. Awaiting reaction from staff.
8526	WAIT_NPR_ERR_GUICL		npError received. Awaiting reaction from staff.
8550	WAIT_NP_REJ_GUI	npCreate	npCreate rejected by donor. Cause for rejection is found in ”message”. Awaiting reaction from staff.
9026	FATAL_WAIT_GUICL		Fatal error. Awaiting reaction from staff.
9127	TERM_BY_CANCEL	npCreate **We are Donor**	npCancel received Awaiting reaction from staff.
9999	X_DONE		Transaction done. Waiting to be archived.

Passive states.

The following ”state” values might be returned from API and should be ignored. They represent an internal (temporary) state:

Value	State	Category	Description
220	DO_RANGE_UPD		NP_Upd Recieved from OCH.
230	SND_NP_UPD_COMPL		NP_Upd_Compl should be sent to OCH
2112	NP_UPD_SNDCPL		Do NP Update (in Database)
2624	CC_WAIT_NP_UPD_COMPL		NP_Compl sent to OCH, wait for all including donor and us
8025	WAIT_NP_UPD_GUICL		NP_Compl recived from all
8125	DONE_NP_UPD_SNDCPL		NP Update DONE
8225	WAIT_RU_UPD_GUICL		NP_Upd_Compl recived from all
8426	WAIT_CHG_ERR_GUICL		NP_Error recieved from OCH
8450	WAIT_NP_ERR_GUI		NP_???? error/rejected by OCH
8526	WAIT_NPR_REJ_GUICL		NP_Reject recieved from OCH
9027	TERM_BY_CANCEL		NP_Cancel send Terminate FLOW...
8620	CC_NP_CREATE_ERR		Error in handling recieved cc:NP_Create

API Reject codes

These codes may be used in conjunction with npReject – reject of porting requests with us as donor.

Reject code	Errortext responded to recipient	Notes
330	The number type II configuration does not match donor's registration	
338	Telephone number not located at donor operator	
339	The Customer ID does not match the telephone number	
349	The telephone number is not active at the donor operator	
350	The telephone number address is undefined	
351	Rejected due to pending change of telephone number	
352	The telephone has pending reactivate order	
353	Rejected due to pending change of customer	
354	Rejected due to pending special terminate order (both telephone term.-old debt term.)	
355	The customer rejects porting (harassment blocking active)	
356	Rejected, donor operator is the customer	
376	Written termination not received by Donor within timeframe	
377	Time to RequestedExecutionDate or possible earliest date exceeds the limit	
378	Network Operator rejects porting Request. Contact Network Operator for reason	
380	The Claimant of the porting is not the subscriber of the telephone number	
381	More than one mobile telephone number in the porting	
382	ICC number does not match telephone number (Prepaid)	
383	Number is either OPS or ERMES	

API Error codes and description

These error codes may be returned in answers' from the API. Please look to respective functions to see which error codes may be present in the various replies from functions.

Error code	Errortext returned by API	Notes
1	”Telephone number is not numeric”	
2	”Telephone number is not 8 ciphers”	
3	”Telephone number is invalid”	
4	”The second telephone number is less than the first telephone number in series	
5	”Not found among service- or network operators”	
6	”Not found among serviceoperators”	
7	”Not registered as a network operator”	
8	”Was not found among network operators”	
9	”Not a valid date format (YYYYMMDD)”	
10	”Date is not numeric (YYYYMMDD)”	
11	”Date is in the past”	
12	”Error in time format (HHMM)”	
13	”SeriesCount is wrongly calculated”	
14	”We are already operator of the number”	
15	”Number is present in another open flow”	
16	”Order number already in use”	
17	”Series format incorrect (hgfedcba- hgfedcba)”	
18	”We are not operator on the number”	
30	”No Portingsets found in DataBase”	
32	”The given PortingSet is not found in database”	
40	”Series missing (SeriesCount and number of given series mismatch)”	
50	”originating ordernumber was not found as an active transaction”	

52	” Originating Ordernumber was not found in the list of parameters”	
54	”Telephonenumber was not found in the list of parameters”	
56	”TA_type was not found in the list of parameters”	
58	”Invalid TA_type”	
60	”Either telephone number or originatingordernumber must be passed”	
62	”Telephone number not found among active transactions”	
64	”More than one transaction found, must be handled manually”	
66	”transaction is not in a state that allow npCancel”	
70	”No transactions found in the state requested”	
80	”Either telephone number, OCHOrderNumber or OriginatingOrderNumber must be passed”	
82	” Parent transaction not found”	
83	”In re-confirm, the confirmed date must be later than the requested.”	
84	”ConfirmationStatus must be entered if confirmed date is different from requested”	
86	”ConfirmationStatus must be a value between 1 and 4”	
87	”ConfirmedExecutionDate is in the past”	
88	” ConfirmedExecutionDate is more than 30 days in the future from requested”	
89	”ConfirmedExecutionDate may not be earlier than RequestedExecutionDate”	
92	”Reject code missing”	
94	”Reject code not found among valid codes”	
General error codes (on socket level)		
500	“bad function call”	
501	“DumpRead in progress”	
1000	“Error in parsed XML”	

Simple test client in PHP

The following code can be used as a simple test of the API functions.

```
<?php

		ini_set('soap.wsdl_cache_enabled', '0');
		ini_set('soap.wsdl_cache_ttl', '0');


		$ConnectOptions = array (
					'login' => 'test_user',
					'password' => 'test_pass',
					'trace' => 0,
					'cache_wsdl' => 1
					);

		$WSDL = "http://localhost:8080?wsdl";

		$apiClient = new SoapClient($WSDL, $ConnectOptions);




		// *****************************************************************
		// about
		// *****************************************************************
		function about() {
			global $apiClient;

			$result = $apiClient->about();
			print_r($result);
		}



		// *****************************************************************
		// apiServerTime
		// *****************************************************************
		function apiservertime() {
			global $apiClient;

			$result = $apiClient->apiServerTime();
			print_r($result);
		}


		// *****************************************************************
		// releaseInfo
		// *****************************************************************
		function releaseInfo() {
			global $apiClient;

			$result = $apiClient->releaseInfo();
			print_r($result);
		}


		// *****************************************************************
		// psetLookup
		// *****************************************************************
		function psetLookup() {
			global $apiClient;

			$result = $apiClient->psetLookup();
			print_r($result);
		}


		// *****************************************************************
		// pnoLookup
		// *****************************************************************
		function pnoLookup($NUM) {
			global $apiClient;

			$result = $apiClient->pnoLookup($NUM);   // phoneNumber is mandatory
			print_r($result);
		}



		// *****************************************************************
		// cpsLookup
```

```
// ************************************************************
function cpsLookup($CPS) {
	global $apiClient;

	$result = $apiClient->cpsLookup($CPS);   // cps is mandatory
	print_r($result);
}



// ************************************************************
// npCreate
// ************************************************************
function npCreate($NUM) {
	global $apiClient;

	$SERIES = array();
	$SERIES[] = array(   'firstnumber' => "56574016", 'lastnumber' => "56574018");
	$SERIES[] = array(   'firstnumber' => "56574439", 'lastnumber' => "56574421");

	$PARAM = array (
		'phoneNumber' => $NUM,            // phoneNumber is mandatory
//		'portingSet' => "TeleCom2",       // optional, if parameter defined AND
                                          // portingSet exist, it will override
                                          // recipientServiceOperator,
                                          // recipientNetworkOperator, spc, municipality,
                                          // routingInfo and chargingInfo
//		'routingInfo' => "33445566",      // optional, if parameter defined it will
                                          // override value
//		'chargingInfo' => "99999999",     // optional, override as above
//		'spc' => "98765",                 // optional, override as above
//		'municipality' => "55555",        // optional, override as above
//		'recipientNetworkOperator' => "01096",   // optional, override as above
//		'recipientServiceOperator' => "01096",   // optional, override as above
//		'currentServiceOperator' => "01081",     // optional, override as above
//		'currentNumberType' => "FIXED",   // optional, override as above
//		'pointOfConnection' => "DONOR",   // optional, override as above
//		'icc' => "01234567890123456678",  // optional, override as above
//		'customerId' => "C56574016",      // optional, override as above
		'requestedExecutionDate' => "20110722",  // optional
//		'requestedExecutionTime' => "1500",      // optional
		'seriesCount' => "2",             // optional, if series exist it must be defined, and
                                          // contain the counts of series parsed in series
		'series' => $SERIES               // optional, see above
	);

	$result = $apiClient->npCreate($PARAM);
	print_r($result);
}



// ************************************************************
// npChange
// ************************************************************
function npChange($NUM) {
	global $apiClient;

	$SERIES = array();
	$SERIES[] = array(   'firstnumber' => "56574016", 'lastnumber' => "56574018");
	$SERIES[] = array(   'firstnumber' => "56574419", 'lastnumber' => "56574421");

	$PARAM = array (
			'phoneNumber' => $NUM,      // phoneNumber is mandatory
			'routingInfo' => "33445566",      // optional, if parameter defined it will
                                              // override current value
//			'chargingInfo' => "99999999",     // optional, override as above
//			'spc' => "98765",                 // optional, override as above
//			'municipality' => "55555",        // optional, override as above
//			'recipientNetworkOperator' => "01096",   // optional, override as above
//			'currentNetworkOperator' => "01074",     // optional, override as above
//			'currentServiceOperator' => "01086",     // optional, override as above
			'newNumberType' => "GSM",         // optional, override as above
//			'portingCase' => "NonPorted",     // optional, override as above
//			'numberPorted' => "Y",            // optional, override as above
			'seriesCount' => "2",             // optional, if series exist it must be
                                              // defined, and contain the counts of series
                                              // parsed in series
			'series' => $SERIES               // optional, see above
	);

	$result = $apiClient->npChange($PARAM);
	print_r($result);
}
```

```
// ************************************************************
// npReturn
// ************************************************************
function npReturn($NUM) {

	global $apiClient;

	$SERIES = array();
	$SERIES[] = array(  'firstnumber' => "56574016", 'lastnumber' => "56574018");
	$SERIES[] = array(  'firstnumber' => "56574419", 'lastnumber' => "56574421");

	$PARAM = array (
			'phoneNumber' => $NUM,     // phoneNumber is mandatory
			'returnDate' => "20120210",// optional, if not defined, number is returned now
			'seriesCount' => "2",      // optional, if series exist it must be defined, and
			                           // contain the counts of series parsed in series
			'series' => $SERIES        // optional, see above
	);

	$result = $apiClient->npReturn($PARAM);
	print_r($result);
}



// ************************************************************
// npCancel
// ************************************************************
function npCancel($NUM) {

	global $apiClient;

	$PARAM = array (    'phoneNumber' => $NUM,     // either phoneNumber OR originatingOrderNumber
	                                               // is mandatory but NOT both
	                    'originatingOrderNumber' => "010100003912"
	);

	$result = $apiClient->npCancel($PARAM);
	print_r($result);
}


// ************************************************************
// npReject
// ************************************************************
function npReject($NUM) {
	global $apiClient;

	$PARAM = array (    'phoneNumber' => $NUM,     // phoneNumber, originatingOrderNumber OR
	                                               // ochOrderNumber is mandatory, but ONLY
	                                               // one of the must be defined
//	                    'originatingOrderNumber' => "010100003912"
//	                    'ochOrderNumber' => "200056144",
	                    'rejectCode' => "355"      // rejectCode is mandatory
	);

	$result = $apiClient->npReject($PARAM);
	print_r($result);
}



// ************************************************************
// npConfirm
// ************************************************************
function npConfirm($NUM) {
	global $apiClient;

	$PARAM = array (
			'phoneNumber' => $NUM,     // phoneNumber, originatingOrderNumber OR
			                           // ochOrderNumber is mandatory, but ONLY
			                           // one of the must be defined
//			'originatingOrderNumber' => "010100003912",
//			'ochOrderNumber' => "200056144",
			'directoryInfo' => 2,      // optional, special handling of phonebook entry
			'confirmedExecutionDate' => "20110728",  // optional, if not defined accept
			                                         // requestedExecutionDate
			'confirmationStatus' => 3  // optional, mandatory if not requestedExecutionDate
			                           // is accepted
	);

	$result = $apiClient->npConfirm($PARAM);
	print_r($result);
}
```

```
// ************************************************************
// ooStatus
// ************************************************************
function ooStatus($NUM) {
	global $apiClient;

	$PARAM = array (
			'originatingOrderNumber' => $NUM, // mandatory
			'recordOffset' => 0		// optional, used if query return > 100 records
	);

	$result = $apiClient->ooStatus($PARAM);
	print_r($result);
}



// ************************************************************
// pnoStatus
// ************************************************************
function pnoStatus($NUM) {
	global $apiClient;

	$PARAM = array (
			'phoneNumber' => $NUM,		// mandatory
			'recordOffset' => 0		// optional, used if query return > 100 records
	);

	$result = $apiClient->pnoStatus($PARAM);
	print_r($result);
}



// ************************************************************
// taStatus
// ************************************************************
function taStatus($TA) {
	global $apiClient;

	$PARAM = array (
			'taType' => $TA,		// mandatory
			'recordOffset' => 0		// optional, used if query return > 100 records
	);

	$result = $apiClient->taStatus($PARAM);
	print_r($result);
}






print_r($apiClient->__getTypes());
print_r($apiClient->__getFunctions());


about();				print "\n";

apiservertime();			print "\n";

releaseInfo();				print "\n";

psetLookup();				print "\n";

pnoLookup("99992699");			print "\n";		// Invalid
pnoLookup("56574016");			print "\n";		// OK

cpsLookup("61011");			print "\n";		// Non existing
cpsLookup("01011");			print "\n";		// TDC

npCreate("42222223");			print "\n";
npCreate("99992699");			print "\n";		// Bad number

npChange("32222223");			print "\n";
npChange("99992699");			print "\n";		// Bad number
```

```
npReturn("32222223");                   print "\n";
npReturn("99992699");                   print "\n";


npCancel("36457595");                   print "\n";

npReject("86941985");                   print "\n";

npConfirm("44351314");                  print "\n";
npConfirm("35351211");                  print "\n";


ooStatus("01010000399112");             print "\n";

pnoStatus("44351314");                  print "\n";

taStatus("AllOpen");                    print "\n";

?>
```