Usage Examples

Before going through the entire document, please have a look at the most frequently asked questions posted by VoipNow Professional SystemAPI users on our forum; we have grouped their questions in this section and came up with an answer to all of them. Therefore, the solution to your problem might already be here!

How can I add incoming call rules?

```
<?php
/* Disable cache */
ini_set('soap.wsdl_cache_enabled', 0);

/* Configure variables */

/* IP of the VoipNow server (including port) */
$IP = "192.168.14.30"; //CHANGE

/* Version of the VoipNow product */
$VERSION = '3.0.0'; //CHANGE

/* Authentication token*/
$TOKEN= '1|n_xOgmCNHc_KLnvOh0T_N8yGM8sX.p05|1|B~GqASQvCd.dpq6tjtV.dmkzk6pX~Dnp'; //CHANGE

/* Extended number of the extension you want to add incoming call rules to */
$EXTENDEDNUMBER = '0007*001'; //CHANGE

/* Id of the time interval you want to assign to the rule
 * the available ids are the actions of the options of the select which allows you to choose a time interval in
the VoipNow interface (from Extension page >> Incoming call rules) */
$INTERVALID = '111'; //CHANGE

/* set SSL context options */
$streamContext = stream_context_create(array(
    'ssl' => array(
        'verify_peer' => false,
        'verify_peer_name' => false,
        'allow_self_signed' => true
     )
));

/* Create SOAP client based on WSDL, with trace for debugging */
$client = new SoapClient('https://' . $IP . '/soap2/schema/' . $VERSION . '/voipnowservice.wsdl', array('trace'
=> 1, 'exceptions' => 1, 'stream_context' => $streamContext));

/* Create authentication headers */
$auth = new stdClass();
$auth->accessToken= $TOKEN;

$userCredentials = new SoapVar($auth, SOAP_ENC_OBJECT, 'http://4psa.com/HeaderData.xsd/' . $VERSION);

$header = new SoapHeader('http://4psa.com/HeaderData.xsd/' . $VERSION, 'userCredentials', $userCredentials,
false);

$client->__setSoapHeaders(array($header));

/**
 * This function computes a random search report for properties like: 'flow', 'type', 'disposion', 'hangupCause'
 *
 * @param numeric $intervalID time interval ID
 * @param numeric $step how many groups of records can be added once. should be at least 1
 *
 * @return array a search query array
 */
function computeCallRules($intervalID, $step = 1) {
    $actions = array('busy', 'congestion', 'hangup', 'transfer', 'cascade', 'authenticate', 'setCallPriority');

    //Rule matches the criteria: o for Match: 0, -1 for Does not match, 1 for Is anonymous and 2 for Is any.
Default value: 0
    $matches = array('0' => '0', //Match
                     '1' => '-1', //Does not match
                     '2' => '1', //Is anonymous
```

```
                    '3' => '2', //Is any
                   );
    //Extension status. 0 for Does not matter, 1 for Registered, -1 for Not Registered.
    $extensionStatuses = array('0' => '0', //Does not matter
                               '1' => '1', //Registered
                               '2' => '-1', // Not Registered.
                              );

    $call_rules = array();
    if ($step >= 1) {
        for($i = 0; $i < $step; $i++) {
            foreach ($actions as $action) {
                $_rule =  array(
                                'match' => $matches[rand(0,3)],
                                'number' => rand(10000,1000000),
                                'position' => '2',
                                'key' => rand(10,100),
                                'intervalID' => $intervalID,
                                );

                switch ($action) {
                    case 'transfer':
                        $_rule += array(
                                        'toNumbers' => array(
                                                            'transferNumber' => '11111',
                                                            'ring' => rand(3,59),
                                                            'call' => rand(0,1),
                                                            'askForCaller' => rand(0,1),
                                                            'transferFromCallee' =>(rand(3,100)%2 == 1) ? 1:0,
                                                            ),
                                        /*
                                        'toVoicemail' => array(
                                                            'transferNumber' => array('0003*001'),
                                                            ),
                                        */
                                        'callStatus' => rand(0,3), //Call status. 0 for Does not matter, 1 for
Not Answered, 2 for Rejected, 3 for Busy
                                        'extensionStatus' => $extensionStatuses[rand(0,2)],
                                        'final' => rand(0,1),
                                    );
                    break;

                    case 'cascade':
                    $_rule += array(
                                    'toNumbers' => array(
                                                        'number' => '2222',
                                                        'ringAfter' => '22',
                                                        ),
                                    'ring' => rand(3,59),
                                    'final' => rand(0,1),
                                    );
                     break;

                    case 'authenticate':
                    $_rule = array(
                                    'password' => '111',
                                    'soundID' => '1112222222222',
                                    'final' => '1',
                                );
                    break;

                    case 'setCallPriority':
                    $_rule = array(
                                   'priority' => '1',
                                );
                     break;

                    default:
                    break;
                }
```

```
                $call_rules[][$action] = $_rule;
            }
        }
    }

    return $call_rules;
}

try {
    $callRule = array(
        'extendedNumber' => $EXTENDEDNUMBER,
        'rule' => computeCallRules($INTERVALID)
    );
   $result = $client->AddCallRulesIn($callRule);
  // $result = $client->GetTimeIntervals(array('ID' => 1));
} catch (SoapFault $exception) {
    echo "ERROR: <br/>" . $exception->getMessage() . "<br/><br/>";
}

echo "REQUEST:<br/>" . htmlentities($client->__getLastRequest()) . "<br/><br/>";
echo "RESPONSE:<br/>" . htmlentities($client->__getLastResponse()) . "<br/>";
?>
```

How do I use the Recharge/GetRecharges methods?

```
<?php
/* Disable cache */
ini_set('soap.wsdl_cache_enabled', 0);

/* Configure variables */

/* IP of the VoipNow server (including port) */
$IP = "192.168.14.30"; //CHANGE

/* Version of the VoipNow product */
$VERSION = '3.0.0'; //CHANGE

/* Authentication token*/
$TOKEN= '1|n_xOgmCNHc_KLnvOh0T_N8yGM8sX.p05|1|B~GqASQvCd.dpq6tjtV.dmkzk6pX~Dnp'; //CHANGE

/* Extended number of the extension you want to add incoming call rules to */
$USERID = '390'; //CHANGE

/* Type of the billing plan: 'credit' for prepaid billing plan, 'limit' for postpaid billing plan */
$TYPE = 'credit';

/* set SSL context options */
$streamContext = stream_context_create(array(
    'ssl' => array(
        'verify_peer' => false,
        'verify_peer_name' => false,
        'allow_self_signed' => true
     )
));

/* Create SOAP client based on WSDL, with trace for debugging */
$client = new SoapClient('https://' . $IP . '/soap2/schema/' . $VERSION . '/voipnowservice.wsdl', array('trace'
=> 1, 'exceptions' => 1, 'stream_context' => $streamContext));

/* Create authentication headers */
$auth = new stdClass();
$auth->accessToken= $TOKEN;

$userCredentials = new SoapVar($auth, SOAP_ENC_OBJECT, 'http://4psa.com/HeaderData.xsd/' . $VERSION);

$header = new SoapHeader('http://4psa.com/HeaderData.xsd/' . $VERSION, 'userCredentials', $userCredentials,
false);

$client->__setSoapHeaders(array($header));
```

```
/**
 * Build a recharge pachage based on the charging plan type
 *
 * @param string $type the charging plan type
 *
 * @return array a search query array
 */
function computeRecharge($type){
    $recharge = array();
    switch ($type) {
        case 'credit':
            $recharge = array (
                                'creditIn' => '3.2', //Increase credit for all incoming calls with
                                'creditOut' => '4',  //Increase credit for all outgoing calls with
                                'orderNo' => 'XN-123/22-02-2011', //Order number
                    );
            break;
        case 'limit':
            $recharge = array (
                                'limitIn' => '1.3', //Increase limit for calls from public network with {} USD
                                'limitOut' => '20', //Increase limit for calls to public network with  {} USD
                                'overusage' => '30', //Increase limit for calls to public network with  {}
minutes
                                'orderNo' => 'XN-123/22-02-2011', //Order number
                                //'monthly' => true,
                    );
            break;
        default:
            echo "TYPE parameter must be 'credit' or 'limit'";
            exit;
    }

    return $recharge;
}

try {
    $recharge = array(
                    'userID' => $USERID,
                    $TYPE => computeRecharge($TYPE)
    );
    $result = $client->Recharge($recharge);

    echo "====Recharge=== REQUEST:<br/>" . htmlentities($client->__getLastRequest()) . "<br/><br/>";
    echo "====Recharge=== RESPONSE:<br/>" . htmlentities($client->__getLastResponse()) . "<br/>";

    //Call GetRecharges method to see all recharges performed for an user
    $result = $client->GetRecharges(array('userID' => $USERID));

    echo "<br/><br/>";
    echo "====GetRecharges=== REQUEST:<br/>" . htmlentities($client->__getLastRequest()) . "<br/><br/>";
    echo "====GetRecharges=== RESPONSE:<br/>" . htmlentities($client->__getLastResponse()) . "<br/>";
} catch (SoapFault $exception) {
    echo "ERROR: <br/>" . $exception->getMessage() . "<br/><br/>";
    echo "REQUEST:<br/>" . htmlentities($client->__getLastRequest()) . "<br/><br/>";
    echo "RESPONSE:<br/>" . htmlentities($client->__getLastResponse()) . "<br/>";
}
?>
```

How do I assign public phone numbers to users?

```
<?php
/* Disable cache */
ini_set('soap.wsdl_cache_enabled', 0);

/* Configure variables */

/* IP of the VoipNow server (including port) */
$IP = "192.168.14.30"; //CHANGE
```

```
/* Version of the VoipNow product */
$VERSION = '3.0.0'; //CHANGE

/* Authentication token*/
$TOKEN= '1|n_xOgmCNHc_KLnvOh0T_N8yGM8sX.p05|1|B~GqASQvCd.dpq6tjtV.dmkzk6pX~Dnp'; //CHANGE

/* Id of the account you want to assign public phone numbers to */
$USERID = '6'; //CHANGE

/* set SSL context options */
$streamContext = stream_context_create(array(
    'ssl' => array(
        'verify_peer' => false,
        'verify_peer_name' => false,
        'allow_self_signed' => true
     )
));

/* Create SOAP client based on WSDL, with trace for debugging */
$client = new SoapClient('https://' . $IP . '/soap2/schema/' . $VERSION . '/voipnowservice.wsdl', array('trace'
=> 1, 'exceptions' => 1, 'stream_context' => $streamContext));

/* Create authentication headers */
$auth = new stdClass();
$auth->accessToken= $TOKEN;

$userCredentials = new SoapVar($auth, SOAP_ENC_OBJECT, 'http://4psa.com/HeaderData.xsd/' . $VERSION);

$header = new SoapHeader('http://4psa.com/HeaderData.xsd/' . $VERSION, 'userCredentials', $userCredentials,
false);

$client->__setSoapHeaders(array($header));

try {
    /* Only the available public phone numbers can be assigned.
       Use GetPublicNoPoll to get available phone numbers for the desired user.
    */
    $result = $client->GetPublicNoPoll(array('userID' => $USERID));

    echo "====GetPublicNoPoll=== REQUEST:<br/>" . htmlentities($client->__getLastRequest()) . "<br/><br/>";
    echo "====GetPublicNoPoll=== RESPONSE:<br/>" . htmlentities($client->__getLastResponse()) . "<br/>";

    /* Use the returned result to assign the available DIDs */
    $publicno = $result->publicNo;
    if (isset($publicno->available)) {
        $available = $publicno->available;
        if (count($available) > 0) {
            $result = $client->AssignPublicNo(array('userID' => $USERID,
                                                    'didID' => $available[ rand(0, (count($available)-1) )]-
>ID));

            echo "====AssignPublicNo=== REQUEST:<br/>" . htmlentities($client->__getLastRequest()) . "<br/><br
/>";
            echo "====AssignPublicNo=== RESPONSE:<br/>" . htmlentities($client->__getLastResponse()) . "<br/>";
        }
    }
} catch (SoapFault $exception) {
    echo "ERROR: <br/>" . $exception->getMessage() . "<br/><br/>";
    echo "REQUEST:<br/>" . htmlentities($client->__getLastRequest()) . "<br/><br/>";
    echo "RESPONSE:<br/>" . htmlentities($client->__getLastResponse()) . "<br/>";
}
?>
```

How do I add a charging plan?

```
<?php
/* Disable cache */
ini_set('soap.wsdl_cache_enabled', 0);

/* Configure variables */
```

```
/* IP of the VoipNow server (including port) */
$IP = "192.168.14.30"; //CHANGE

/* Version of the VoipNow product */
$VERSION = '3.0.0'; //CHANGE

/* Authentication token*/
$TOKEN= '1|n_xOgmCNHc_KLnvOh0T_N8yGM8sX.p05|1|B~GqASQvCd.dpq6tjtV.dmkzk6pX~Dnp'; //CHANGE

/* set SSL context options */
$streamContext = stream_context_create(array(
    'ssl' => array(
        'verify_peer' => false,
        'verify_peer_name' => false,
        'allow_self_signed' => true
     )
));

/* Create SOAP client based on WSDL, with trace for debugging */
$client = new SoapClient('https://' . $IP . '/soap2/schema/' . $VERSION . '/voipnowservice.wsdl', array('trace'
=> 1, 'exceptions' => 1, 'stream_context' => $streamContext));

/* Create authentication headers */
$auth = new stdClass();
$auth->accessToken= $TOKEN;

$userCredentials = new SoapVar($auth, SOAP_ENC_OBJECT, 'http://4psa.com/HeaderData.xsd/' . $VERSION);

$header = new SoapHeader('http://4psa.com/HeaderData.xsd/' . $VERSION, 'userCredentials', $userCredentials,
false);

$client->__setSoapHeaders(array($header));

/**
 * Build a random charging plan package
 *
 * @return array charging plan data
 */
function computeChargingPlan() {
    $plan = array('postpaid', 'prepaid');
    $charge = array('inherit', 'fixed');

    $charging_plan = array(
        'name' => 'SytemAPI Charging Plan ' . time() . "_" . rand(1, 100),
        'default' => '0',
        'channelRuleID' => '1',
        'allowIn' => '1', //Allow incoming calls
        'allowOut' => '1', //Allow calls to public network
        'allowLocal' => '1', //Allow local calls to extensions owned by the same client like caller
        'allowExtended' => '1' //Allow extended local calls to extensions owned by other clients in the
infrastructure
     );

     $plan_type = $plan[time() % 2];
     $charging_plan['planType'] = $plan_type;   //'postpaid' or 'prepaid'

     if ($plan_type == 'postpaid') {
         $charging_plan += array(
             'includedCreditOut' => array(
                 'unlimited' => false, //for Unlimited set true
                 '_' => 500 //Limit calls to public network to amount
             ),
             'includedCreditIn' => array(
                 'unlimited' => false,
                 '_' => 500 //Limit calls from public network to amount
             ),
             'externalMin' => array(
                 'minutes' => time() % 100, //Limit calls to public network to minutes
                 'intervalID' => 0 // Anytime interval
             )
```

```
        );
    }

    if ($plan_type == 'prepaid') {
        $charging_plan += array(
            'initialCreditOut' => array(
                'unlimited' => false, //Initial credit available for outgoing calls
                '_' => 9
            ),
            'initialCreditIn' => array(
                'unlimited' => false, //Initial credit available for incoming calls
                '_' => 5000
            ),
            'chargeOut' => 2, //Charge outgoing calls indivisible for the first
            'thenChargeOut' => 1, //After the first segment charge every
            'chargeIn' => 2, //Charge incoming calls indivisible for the first
            'thenChargeIn' => 1, //After the first segment charge every
       );
     }

     $chargeMethod = $charge[time() % 2];
     $charging_plan['chargeMethod'] = $chargeMethod;
     if ($chargeMethod == 'fixed') {
         $charging_plan += array(
             'fixedCharge' => array(
                 'local' => rand(1,10)/10, //Charge local calls to extensions
                 'extended' => rand(1,10)/10, //Charge extended local calls to extensions
                 'externalIncoming' => rand(1,10)/10, //Charge incoming calls
                 'external' => array(
                     'charge' => rand(1,10)/10, //Charge outgoing calls
                     'intervalID' => 0
                 )
             )
         );
     }

     if ($chargeMethod == 'inherit') {
         $charging_plan += array(
             'inheritedCharge' => array(
                 'local' => array(
                     'mulFactor' => rand(3,10),  //Charge local calls to extensions
                     'adjustment' => rand(3,10)/10
                 ),
                 'extended' => array(
                     'mulFactor' => rand(3,10),  //Charge extended local calls to extensions
                     'adjustment' => rand(3,10)/10
                 ),
                 'external' => array(
                     'mulFactor' => rand(3,10),  //Charge outgoing calls
                     'adjustment' => rand(3,10)/10
                 ),
                 'externalIncoming' => array(
                     'mulFactor' => rand(3,10),  //Charge incoming calls
                     'adjustment' => rand(3,10)/10
                 ),
             )
         );
    }

    return $charging_plan;
}

try {
    $chargingPlan = computeChargingPlan();
    $result = $client->AddChargingPlan($chargingPlan);
} catch (SoapFault $exception) {
    echo "ERROR: <br/>" . $exception->getMessage() . "<br/><br/>";
}

echo "REQUEST:<br/>" . htmlentities($client->__getLastRequest()) . "<br/><br/>";
```

```
echo "RESPONSE:<br/>" . htmlentities($client->__getLastResponse()) . "<br/>";
?>
```

How can I get calls information for a specific timeframe?

```
<?php
/* Disable cache */
ini_set('soap.wsdl_cache_enabled', 0);

/* Configure variables */

/* IP of the VoipNow server (including port) */
$IP = "192.168.14.30"; //CHANGE

/* Version of the VoipNow product */
$VERSION = '3.0.0'; //CHANGE

/* Authentication token*/
$TOKEN= '1|n_xOgmCNHc_KLnvOh0T_N8yGM8sX.p05|1|B~GqASQvCd.dpq6tjtV.dmkzk6pX~Dnp'; //CHANGE

/* The login(username) of the account you want to find information about */
$USER = 'dclient'; //CHANGE

/* the start date of the report */
$STARTDATE = '2011-01-01'; //CHANGE

/* the end date of the report */
$ENDDATE = '2011-07-31'; //CHANGE

/* set SSL context options */
$streamContext = stream_context_create(array(
    'ssl' => array(
        'verify_peer' => false,
        'verify_peer_name' => false,
        'allow_self_signed' => true
     )
));

/* Create SOAP client based on WSDL, with trace for debugging */
$client = new SoapClient('https://' . $IP . '/soap2/schema/' . $VERSION . '/voipnowservice.wsdl', array('trace'
=> 1, 'exceptions' => 1, 'stream_context' => $streamContext));

/* Create authentication headers */
$auth = new stdClass();
$auth->accessToken= $TOKEN;

$userCredentials = new SoapVar($auth, SOAP_ENC_OBJECT, 'http://4psa.com/HeaderData.xsd/' . $VERSION);

$header = new SoapHeader('http://4psa.com/HeaderData.xsd/' . $VERSION, 'userCredentials', $userCredentials,
false);

$client->__setSoapHeaders(array($header));


/**
 * This function computes a random search report for properties like: 'flow', 'type', 'disposion', 'hangupCause'
 *
 * @return array a search query array
 */
function getSearchQuery() {
    $flow = array('in', 'out');
    $type = array('local', 'elocal', 'out'); //call context
    $disposion = array('ANSWERED', 'BUSY', 'FAILED', 'NO ANSWER', 'UNKNOWN', 'NOT ALLOWED'); //Call ended with
    $hangupCause = array('1',     // Unallocated (1)
                         '2',     // No Route to Network (2)
                         '3',     // No Route to Destination (3)
                         '6',     // Channel Unacceptable (6)
                         '7',     // Call Awarded Delivered (7)
                         '16',    // Normal Clearing (16)
                         '17',    // User Busy (17)
```

```
                        '18',    // No User Response (18)
                        '19',    // No Answer (19)
                        '20',    // Subscriber Absent (20)
                        '21',    // Call Rejected (21)
                        '22',    // Number Changed (22)
                        '27',    // Destination Out of Order (27)
                        '28',    // Invalid Number Format (28)
                        '29',    // Facility Rejected (29)
                        '30',    // Response to Status Enquiry (30)
                        '31',    // Normal Unspecified (31)
                        '34',    // Normal Circuit Congestion (34)
                        '38',    // Network Out of Order (38)
                        '41',    // Normal Temporary Failure (41)
                        '42',    // Switching Equipment Congestion (42)
                        '43',    // Access Info Discarded (43)
                        '44',    // Requested Channel Unavailable (44)
                        '45',    // Pre Empted (45)
                        '50',    // Facility Not Subscribed (50)
                        '52',    // Outgoing Call Barred (52)
                        '54',    // Incoming Call Barred (54)
                        '57',    // Bearer Capability Not Authorized (57)
                        '58',    // Bearer Capability Not Available (58)
                        '65',    // Bearer Capability Not Implemented (65)
                        '66',    // Channel Not Implemented (66)
                        '69',    // Facility Not Implemented (69)
                        '81',    // Invalid Call Reference (81)
                        '88',    // Incompatible Destination (88)
                        '95',    // Invalid Message Unspecified (95)
                        '96',    // Mandatory Information Element Missing (96)
                        '97',    // Message Type Nonexistent (97)
                        '98',    // Wrong Message (98)
                        '99',    // Information Element Nonexistent (99)
                        '100',   // Invalid Information Element Contents (100)
                        '101',   // Wrong Call State (101)
                        '102',   // Recovery on Timer Expire (102)
                        '103',   // Mandatory Information Element Length Error (103)
                        '111',   // Protocol Error (111)
                        '127');  // Interworking (127)

 return array('flow' => $flow[time() % 2],
              'type' => $type[time() % 3],
              'disposion' => $disposion[time() % 6],
             'hangupCause' => $hangupCause[rand(0,44)]);
}

try {
    $report_query = getSearchQuery();
    $report = array(
        'login' => $USER,
        'interval' => array(
        'startDate' => $STARTDATE,
        'endDate' => $ENDDATE),
        'flow' => $report_query['flow'],
        'type' => $report_query['type'],
        'disposion' => $report_query['disposion'],
        'hangupCause' => $report_query['hangupCause'],
        'records' => '100',
    );
    $result = $client->CallReport($report);
} catch (SoapFault $exception) {
    echo "ERROR: <br/>" . $exception->getMessage() . "<br/><br/>";
}

echo "REQUEST:<br/>" . htmlentities($client->__getLastRequest()) . "<br/><br/>";
echo "RESPONSE:<br/>" . htmlentities($client->__getLastResponse()) . "<br/>";

?>
```

How can I get call cost information?

```
<?php
/* Disable cache */
ini_set('soap.wsdl_cache_enabled', 0);

/* Configure variables */

/* IP of the VoipNow server (including port) */
$IP = "192.168.14.30"; //CHANGE

/* Version of the VoipNow product */
$VERSION = '3.0.0'; //CHANGE

/* Authentication token*/
$TOKEN= '1|n_xOgmCNHc_KLnvOh0T_N8yGM8sX.p05|1|B~GqASQvCd.dpq6tjtV.dmkzk6pX~Dnp'; //CHANGE

/* Id of the account you want to report the costs */
$USERID = '4'; //CHANGE

/* Start date of the time interval you want cost information */
$STARTDATE = '2011-07-01'; //CHANGE

/* End date of the time interval you want cost information */
$ENDDATE = '2011-07-31'; //CHANGE

/* set SSL context options */
$streamContext = stream_context_create(array(
    'ssl' => array(
        'verify_peer' => false,
        'verify_peer_name' => false,
        'allow_self_signed' => true
     )
));

/* Create SOAP client based on WSDL, with trace for debugging */
$client = new SoapClient('https://' . $IP . '/soap2/schema/' . $VERSION . '/voipnowservice.wsdl', array('trace'
=> 1, 'exceptions' => 1, 'stream_context' => $streamContext));

/* Create authentication headers */
$auth = new stdClass();
$auth->accessToken= $TOKEN;

$userCredentials = new SoapVar($auth, SOAP_ENC_OBJECT, 'http://4psa.com/HeaderData.xsd/' . $VERSION);

$header = new SoapHeader('http://4psa.com/HeaderData.xsd/' . $VERSION, 'userCredentials', $userCredentials,
false);

$client->__setSoapHeaders(array($header));

try {
    $report = array(
        'userID' => $USERID,
        'interval' => array (
                        'startDate' => $STARTDATE,
                        'endDate' => $ENDDATE),
        /* comment interval and uncomment year and/or month in order to retrieve data for a specified period */
        //'year' => '2011',
        //'month' => '06'
    );

    $result = $client->CallCosts($report);
} catch (SoapFault $exception) {
    echo "ERROR: <br/>" . $exception->getMessage() . "<br/><br/>";
}

echo "REQUEST:<br/>" . htmlentities($client->__getLastRequest()) . "<br/><br/>";
echo "RESPONSE:<br/>" . htmlentities($client->__getLastResponse()) . "<br/>";
?>
```

How can I add several public phone numbers at once?

```
<?php
/* Disable cache */
ini_set('soap.wsdl_cache_enabled', 0);

/* Configure variables */

/* IP of the VoipNow server (including port) */
$IP = "192.168.14.30"; //CHANGE

/* Version of the VoipNow product */
$VERSION = '3.0.0'; //CHANGE

/* Authentication token*/
$TOKEN= '1|n_xOgmCNHc_KLnvOh0T_N8yGM8sX.p05|1|B~GqASQvCd.dpq6tjtV.dmkzk6pX~Dnp'; //CHANGE

/* The id of the channel the numbers are added for */
$CHANGELID = '1'; //CHANGE
/* set SSL context options */
$streamContext = stream_context_create(array(
    'ssl' => array(
        'verify_peer' => false,
        'verify_peer_name' => false,
        'allow_self_signed' => true
     )
));

/* Create SOAP client based on WSDL, with trace for debugging */
$client = new SoapClient('https://' . $IP . '/soap2/schema/' . $VERSION . '/voipnowservice.wsdl', array('trace'
=> 1, 'exceptions' => 1, 'stream_context' => $streamContext));

/* Create authentication headers */
$auth = new stdClass();
$auth->accessToken= $TOKEN;

$userCredentials = new SoapVar($auth, SOAP_ENC_OBJECT, 'http://4psa.com/HeaderData.xsd/' . $VERSION);

$header = new SoapHeader('http://4psa.com/HeaderData.xsd/' . $VERSION, 'userCredentials', $userCredentials,
false);

$client->__setSoapHeaders(array($header));

try {

        $type_array = array('stacked', 'exclusive');

        for ($i = 0; $i < rand(3, 10); $i++){
                $public_no['publicNo'][] = array('channelID' => $CHANGELID ,
                                                                        'phoneNo' => rand(1000000, 10000000),
//Public phone number
                                                                        'did' => rand(1000000,
10000000),        //DID (Direct Inward Dialing) number
                                                                        'location' => 'test_location_'.rand
(10, 100), //Location
                                                                        'cost' => rand(1,10), //Monthly amount
paid to provider
                                                                        'incomingCost' => array('cost' =>rand
(1,10), 'interval' =>rand(10,200)), //Incoming calls cost: {cost} money_unit every {interval} seconds
                                                                        'type' => $type_array[rand(1,100) %
2],        //Assignation method: stacked or exclusive
                                                                );
        }
    $result = $client->AddPublicNo($public_no);
        echo "<pre>====AddPublicNo=== REQUEST:\n</pre>";
        echo preg_replace(array("/></i", "/</i", "/>/i", "/\n/i"), array("&gt;\n&lt;", "&lt;", "&gt;", "<br
/>"), $client->__getLastRequest() );
        echo "<pre>====AddPublicNo=== RESPONSE:\n</pre>";
        echo preg_replace(array("/></i", "/</i", "/>/i", "/\n/i"), array("&gt;\n&lt;", "&lt;", "&gt;", "<br
/>"), $client->__getLastResponse() );
} catch (SoapFault $exception) {
    echo "ERROR: <br/>" . $exception->getMessage() . "<br/><br/>";
```

```
    echo "REQUEST:<br/>" . htmlentities($client->__getLastRequest()) . "<br/><br/>";
    echo "RESPONSE:<br/>" . htmlentities($client->__getLastResponse()) . "<br/>";
}
?>
```

How do I use the SetupExtension method?

```
<?php
/* Disable cache */
ini_set('soap.wsdl_cache_enabled', 0);

/* Configure variables */

/* IP of the VoipNow server (including port) */
$IP = "192.168.14.30"; //CHANGE

/* Version of the VoipNow product */
$VERSION = '3.0.0'; //CHANGE

/* Authentication token*/
$TOKEN= '1|n_xOgmCNHc_KLnvOh0T_N8yGM8sX.p05|1|B~GqASQvCd.dpq6tjtV.dmkzk6pX~Dnp'; //CHANGE

/* The extended number of the extension you want to setup */
$EXTENDEDNUMBER = '0007*001'; //CHANGE
/* set SSL context options */
$streamContext = stream_context_create(array(
    'ssl' => array(
        'verify_peer' => false,
        'verify_peer_name' => false,
        'allow_self_signed' => true
     )
));

/* Create SOAP client based on WSDL, with trace for debugging */
$client = new SoapClient('https://' . $IP . '/soap2/schema/' . $VERSION . '/voipnowservice.wsdl', array('trace'
=> 1, 'exceptions' => 1, 'stream_context' => $streamContext));

/* Create authentication headers */
$auth = new stdClass();
$auth->accessToken= $TOKEN;

$userCredentials = new SoapVar($auth, SOAP_ENC_OBJECT, 'http://4psa.com/HeaderData.xsd/' . $VERSION);

$header = new SoapHeader('http://4psa.com/HeaderData.xsd/' . $VERSION, 'userCredentials', $userCredentials,
false);

$client->__setSoapHeaders(array($header));

function setup_extension_phone(){
        $phone_terminal = array(

                //Basic Settings
                'mohFolder' => '////',
                'sendCallerID' => '1', //Caller-ID name in public calls. Values: 1- server, 0-equipment, 3-
user. Default: by server.
                //'callerName' => 'My Name',  //Parameter used when 'callerIDSend' is set to user (value = 3).
Default value is set to the extension's name.
                'sendCallerNo' => '1',  //Caller-ID number in public calls. Values: 1- server, 0-equipment, 3-
user. Default: by server.
                //'callerNumber' => '1234567', //Parameter used when 'sendCallerNo' is set to user (value = 3).
Default value is set to the extension's phone number.
                'callerIDRefs' => array('1','2'), //Currently used CallerID numbers; this parameter is optional
and can be set only if 1 - server option is provided for sendCallerNo field.
                                                                                        //this is obtained from
GetPublicNoPoll response

                'defaultCallerIDRef' => '2', //Default callerID number; this parameter is used when
'sendCallerNo' is set to server (value=1).
                'anonymous' => true, // Do not send CallerID on public calls
                'callerIDInternal' => true, // Send public CallerID on internal calls (if public callerID is
```

```
available)
                'callerIDOnTransfer' => true, //Preserve original CallerID on transfered calls
                'SIPIdentity' => true, //Send SIP P-Asserted-Identity header
                'noAnswer' => '75', //Hangup when extension does not answer in X seconds
                'parkTimeout' => '20', //Do not keep calls in parking lots for more than X seconds

                //Instant Messaging

                'IM' => false, //Enable chat server access

                //Calling Features

                'callWaiting' => true, //Call waiting active
                'dnd' => true, //Do not disturb function active
                'dndSndActive' => 1, //Play sound
                'dndSnd' => '374', //sound id

                //Password Protection

                'phoneAccess' => true, //Protected phone access active
                'block' => true, //Restrict access to phone. This option is available only when 'phoneAccess'
is set to true.
                'phoneAccessPassword' => '000', //Password to access telephony
        );
                return $phone_terminal;
}

        try {
                $phoneTerminal = array(
                        'extendedNumber' => $EXTENDEDNUMBER ,
                        'phoneTerminal' => setup_extension_phone()
                );
                $result = $client->SetupExtension($phoneTerminal);
        } catch (SoapFault $exception) {
                echo "ERROR: <br/>" . $exception->getMessage() . "<br/><br/>";
        }

echo "REQUEST:<br/>" . htmlentities($client->__getLastRequest()) . "<br/><br/>";
echo "RESPONSE:<br/>" . htmlentities($client->__getLastResponse()) . "<br/>";

?>
```

How do I use the AddCallRulesOutGroup method to add outgoing routing rules group for a channel?

```
<?php
/* Disable cache */
ini_set('soap.wsdl_cache_enabled', 0);

/* Configure variables */

/* IP of the VoipNow server (including port) */
$IP = "192.168.14.30"; //CHANGE

/* Version of the VoipNow product */
$VERSION = '3.0.0'; //CHANGE

/* Authentication token*/
$TOKEN= '1|n_xOgmCNHc_KLnvOh0T_N8yGM8sX.p05|1|B~GqASQvCd.dpq6tjtV.dmkzk6pX~Dnp'; //CHANGE
/* set SSL context options */
$streamContext = stream_context_create(array(
    'ssl' => array(
        'verify_peer' => false,
        'verify_peer_name' => false,
        'allow_self_signed' => true
     )
));

/* Create SOAP client based on WSDL, with trace for debugging */
$client = new SoapClient('https://' . $IP . '/soap2/schema/' . $VERSION . '/voipnowservice.wsdl', array('trace'
=> 1, 'exceptions' => 1, 'stream_context' => $streamContext));
```

```
/* Create authentication headers */
$auth = new stdClass();
$auth->accessToken= $TOKEN;

$userCredentials = new SoapVar($auth, SOAP_ENC_OBJECT, 'http://4psa.com/HeaderData.xsd/' . $VERSION);

$header = new SoapHeader('http://4psa.com/HeaderData.xsd/' . $VERSION, 'userCredentials', $userCredentials,
false);

$client->__setSoapHeaders(array($header));


function generate_rule($channelID=false, $intervalID=0) {
        //channelID is default best cost
        $prefix_operation = array(0=>'prefix', 1=>'add', 2=>'replace', 3=>'substract');
        $action_array = array(0=>'block', 1 =>'transfer', 2=>'process');
        $i = 0;
        $caller_id = array('prefix' => array(
                                                                'callerIDPrefixOperation'=> 'prefix',
                                                                'callerIDPrefix' => '1',
                                                                'callerIDMatch'=> '2',
                                                                ),
                                                'add' => array(
                                                                'callerIDPrefixOperation'=> 'add',
                                                                'callerIDNumberAdd'=> '3',
                                                                'callerIDDigitsAfterAdd'=> '4',
                                                                'callerIDMatch'=> '5',
                                                                ),

                                                'replace' => array(
                                                                'callerIDPrefixOperation'=> 'replace',
                                                                'callerIDPrefix' => '6',
                                                                'callerIDMatch'=> '7',
                                                                ),
                                                'substract' => array(
                                                                'callerIDPrefixOperation'=> 'substract',
                                                                'callerIDDigits' => '8',
                                                                'callerIDDigitsAfter' => '9',
                                                                'callerIDMatch'=> '10',

),

                                                );

        foreach ($prefix_operation as $k =>$v){
                if($v == 'prefix'){
                        foreach ($action_array as $key =>$val){
                                if($val == 'transfer' or $val == 'process'){
                                        $arr[$i] = array(
                                                'action' =>$val,
                                                'number' => time() % 1000000 . rand(1, 1000),
                                                'intervalID' => $intervalID,
                                                'comingFrom' => '9999',
                                                'channelID' => $channelID,
                                        );
                                        $rest = array(
                                                'prefixOperation' => 'prefix',
                                                'prefix' => '5',
                                                'final' => '',
                                                );
                                        $arr[$i] = array_merge($arr[$i],$rest );
                                        $arr[$i] = array_merge($arr[$i],$caller_id['prefix'] );

                                        $i++;
                                }
                        }
                }
                if($v == 'add'){
                        foreach ($action_array as $key =>$val){
                                if($val == 'transfer' or $val == 'process'){
```

```
                                $arr[$i] = array(
                                        'action' =>$val,
                                        'number' => time() % 1000000 . rand(1, 1000),
                                        'intervalID' => $intervalID,
                                        'comingFrom' => '4444',
                                        'channelID' => $channelID,
                                );
                                $rest3 = array(
                                        'prefixOperation' => 'add',
                                        'numberAdd' =>'2',
                                        'digitsAfterAdd' => '1',
                                        'prefix' => '4',
                                        'final' => '',
                                );
                                $arr[$i] = array_merge($arr[$i],$rest3 );
                                $arr[$i] = array_merge($arr[$i],$caller_id['add'] );
                                $i++;
                        }
                }
        }
        if($v == 'replace'){
                foreach ($action_array as $key =>$val){
                        if($val == 'transfer' or $val == 'process'){
                                $arr[$i] = array(
                                        'action' =>$val,
                                        'number' => time() % 1000000 . rand(1, 1000),
                                        'intervalID' => $intervalID,
                                        'comingFrom' => '8888',
                                        'channelID' => $channelID,
                                );
                                $rest1 = array(
                                        'prefixOperation' => 'replace',
                                        'final' => '',
                                );
                                $arr[$i] = array_merge($arr[$i],$rest1 );
                                $arr[$i] = array_merge($arr[$i],$caller_id['replace'] );
                                $i++;
                        }
                }
        }
        if($v == 'substract'){
                foreach ($action_array as $key =>$val){
                        if($val == 'transfer' or $val == 'process'){
                                $arr[$i] = array(
                                        'action' =>$val,
                                        'number' => time() % 1000000 . rand(1, 1000),
                                        'intervalID' => $intervalID,
                                        'comingFrom' => '555',
                                        'channelID' => $channelID,
                                );
                                $rest2 = array(
                                        'prefixOperation' => 'substract',
                                        'digits' => '3',
                                        'digitsAfter' => '1',
                                        'final' => '',
                                );
                                $arr[$i] = array_merge($arr[$i],$rest2 );
                                $arr[$i] = array_merge($arr[$i],$caller_id['substract'] );
                                $i++;
                        }
                }
        }
}


$arr[$i] = array(
        'action' =>'block',
        'number' => time() % 1000000 . rand(1, 1000),
        'intervalID' => $intervalID,
        'commingFrom' => '4444',
);
```

```
        $i++;
        $arr[$i] = array(
                'action' =>'portability',
                'engine'=>'123123123',
                'number' => time() % 1000000 . rand(1, 1000),
                'intervalID' => $intervalID,
                'commingFrom' => '4444',
        );

        return $arr;
}



class test {
        function test(){
                $this->name = 'My Rule ' . time();
                return $this;
        }
}


if (true)
{
                $obj = new test();
                $obj->rules = generate_rule(); //Add outgoing routing rules group
                $result = $client->AddCallRulesOutGroup($obj);

                echo "<br><br>REQUEST AddCallRulesOutGroup:<br>";
                echo preg_replace(array("/></i", "/</i", "/>/i", "/\n/i"), array("&gt;\n&lt;", "&lt;", "&gt;",
"<br />"), $client->__getLastRequest() );
        echo "<br><br>RESPONSE AddCallRulesOutGroup:<br>";
        echo preg_replace(array("/></i", "/</i", "/>/i", "/\n/i"), array("&gt;\n&lt;", "&lt;", "&gt;", "<br
/>"), $client->__getLastResponse() );
}
?>
```

How do I use the AddCallRulesOutGroup method to add outgoing routing rules group for a user?

```php
<?php
/* Disable cache */
ini_set('soap.wsdl_cache_enabled', 0);

/* Configure variables */

/* IP of the VoipNow server (including port) */
$IP = "192.168.14.30"; //CHANGE

/* Version of the VoipNow product */
$VERSION = '3.0.0'; //CHANGE

/* Authentication token*/
$TOKEN= '1|n_xOgmCNHc_KLnvOh0T_N8yGM8sX.p05|1|B~GqASQvCd.dpq6tjtV.dmkzk6pX~Dnp'; //CHANGE

/* The id of the account the rule is added for */
$USERID = '6'; //CHANGE

/* The id of time interval used by rule*/
$TIMEINTERVALID = '81';
/* set SSL context options */
$streamContext = stream_context_create(array(
    'ssl' => array(
        'verify_peer' => false,
        'verify_peer_name' => false,
        'allow_self_signed' => true
     )
));

/* Create SOAP client based on WSDL, with trace for debugging */
$client = new SoapClient('https://' . $IP . '/soap2/schema/' . $VERSION . '/voipnowservice.wsdl', array('trace'
=> 1, 'exceptions' => 1, 'stream_context' => $streamContext));

/* Create authentication headers */
$auth = new stdClass();
$auth->accessToken= $TOKEN;

$userCredentials = new SoapVar($auth, SOAP_ENC_OBJECT, 'http://4psa.com/HeaderData.xsd/' . $VERSION);

$header = new SoapHeader('http://4psa.com/HeaderData.xsd/' . $VERSION, 'userCredentials', $userCredentials,
false);

$client->__setSoapHeaders(array($header));

try {
        $rule = array(
                        'name' => 'My Rule ' . time(),
                        'userID' => $USERID,
                        'rules' => array(
                                        'action' =>'block',
                                        'number' => time() % 1000000 . rand(1, 1000),
                                        'intervalID' => $TIMEINTERVALID,
                                        'commingFrom' => '4444',
                                ),
                );
        $result = $client->AddCallRulesOutGroup($rule);
    } catch (SoapFault $exception) {
        echo "ERROR: <br/>" . $exception->getMessage() . "<br/><br/>";
    }
    echo "<br><br>REQUEST AddCallRulesOutGroup:<br>";
    echo preg_replace(array("/></i", "/</i", "/>/i", "/\n/i"), array("&gt;\n&lt;", "&lt;", "&gt;", "<br />"),
$client->__getLastRequest() );
    echo "<br><br>RESPONSE AddCallRulesOutGroup:<br>";
    echo preg_replace(array("/></i", "/</i", "/>/i", "/\n/i"), array("&gt;\n&lt;", "&lt;", "&gt;", "<br />"),
$client->__getLastResponse() );

?>
```

How do I get account calls from a time interval?

```
<?php
/* Disable cache */
ini_set('soap.wsdl_cache_enabled', 0);

/* Configure variables */

/* IP of the VoipNow server (including port) */
$IP = "192.168.14.30"; //CHANGE

/* Version of the VoipNow product */
$VERSION = '3.0.0'; //CHANGE

/* Authentication token*/
$TOKEN= '1|n_xOgmCNHc_KLnvOh0T_N8yGM8sX.p05|1|B~GqASQvCd.dpq6tjtV.dmkzk6pX~Dnp'; //CHANGE

/* The id of the account the request is performed for */
$USERID = '6'; //CHANGE
/* set SSL context options */
$streamContext = stream_context_create(array(
    'ssl' => array(
        'verify_peer' => false,
        'verify_peer_name' => false,
        'allow_self_signed' => true
     )
));

/* Create SOAP client based on WSDL, with trace for debugging */
$client = new SoapClient('https://' . $IP . '/soap2/schema/' . $VERSION . '/voipnowservice.wsdl', array('trace'
=> 1, 'exceptions' => 1, 'stream_context' => $streamContext));

/* Create authentication headers */
$auth = new stdClass();
$auth->accessToken= $TOKEN;

$userCredentials = new SoapVar($auth, SOAP_ENC_OBJECT, 'http://4psa.com/HeaderData.xsd/' . $VERSION);

$header = new SoapHeader('http://4psa.com/HeaderData.xsd/' . $VERSION, 'userCredentials', $userCredentials,
false);

$client->__setSoapHeaders(array($header));

try {

        $request_params = array(
                                                        'userID' => $USERID,
                                                        'interval' => array(

'startDate'=>'2011-12-19',

'endDate'=>'2011-12-19',
                                                                                                        )
                                                );

    $result = $client->CallReport($request_params);
        echo "<pre>====CallReport=== REQUEST:\n</pre>";
        echo preg_replace(array("/></i", "/</i", "/>/i", "/\n/i"), array("&gt;\n&lt;", "&lt;", "&gt;", "<br
/>"), $client->__getLastRequest() );
        echo "<pre>====CallReport=== RESPONSE:\n</pre>";
        echo preg_replace(array("/></i", "/</i", "/>/i", "/\n/i"), array("&gt;\n&lt;", "&lt;", "&gt;", "<br
/>"), $client->__getLastResponse() );
} catch (SoapFault $exception) {
    echo "ERROR: <br/>" . $exception->getMessage() . "<br/><br/>";
    echo "REQUEST:<br/>" . htmlentities($client->__getLastRequest()) . "<br/><br/>";
    echo "RESPONSE:<br/>" . htmlentities($client->__getLastResponse()) . "<br/>";
}
?>
```