

Volo API Cookbook

tl;dr - Jump to “Volo + external DB” and “Injection drop - DynQueue” sections for the good stuff, the rest is a beginner friendly intro.

Foreword

API was developed over time to allow people who already manually ran their Volos to offload more and more work onto their scripts. Methods were added upon popular request, one by one.

API comes in naturally but only after you already manually used your volo for a while and know your way around, because your developer will need explanations from either you or me (joe@volomp.com).

API needs the admin password to auth, and customer id to know which data to use. Enter those into the examples, where designated.

Hey dev, I know terms like “ip space”, “easy links”, “throttling_available_ranges” etc. are confusing, so shoot an email to joe@volomp.com for a quick skype/talky.io screenshare. You will see the logic behind those API calls and why there are so many optional args (which translate 1-1 into UI checkboxes).

First we are going to cover the basics, and then the last two bits will be about bulk mailing without use of the on-Volo database, and injecting into a dynamic queue.

Getting our feet wet

What is utils.php?

Part of libXMLRPC library, just a file you require. Attached at the end of this file.

What's user_id?

Run from shell:

```
[user@server ~]$ mysql -u root --batch -e 'select UserID, Username from web.accounts;'
UserID Username
1      chester
2      rob
```

So now you know the user_id of the account you want to use.

What are the mailing lists ids?

Run this php script:

```
<?php
require_once("utils.php");

print_r(
    xu_rpc_http_concise(
        array(
            'method' => 'mailing_lists',
            'args' => array(
                'user_id' => 4, // put your id here
            ),
            'host' => '<YOUR VOLO IP ADDRESS>', // put your Volo server ip here
            'uri' => '/xapi/server.php',
            'port' => 80,
            'user' => 'admin',
            'pass' => '<YOUR ADMIN UI PASSWORD>' // put your Volo admin web ui password here
        )
    )
);
```

```
)  
)  
?>
```

To get output like this:

```
Array  
(  
  [0] => Array  
    (  
      [list_id] => 83  
      [list_name] => mike_aug_IW_clean  
      [recipient_count] => 1919  
      [from] => "PLACEHOLDER" <PLACEHOLDER@%$_sdomain%>  
      [header] => 0  
      [footer] => 6  
      [disabled_delivery] => 0  
    )  
  
  [1] => Array  
    (  
      [list_id] => 82  
      [list_name] => live_scandi  
      [recipient_count] => 1000  
      [from] => "PLACEHOLDER" <PLACEHOLDER@%$_sdomain%>  
      [header] => 0  
      [footer] => 6  
      [disabled_delivery] => 0  
    )  
)
```

So now you know the list_id for each mailing list.

What's the ip space?

Run this php script:

```
<?php
require_once("utils.php");

print_r(
    xu_rpc_http_concise(
        array(
            'method' => 'get_ip_space',
            'args' => array(
                'user_id' => 4,
            ),
            'host' => '<YOUR VOLO IP ADDRESS>', // put your Volo server ip here
            'uri' => '/xapi/server.php',
            'port' => 80,
            'user' => 'admin',
            'pass' => '<YOUR ADMIN UI PASSWORD>' // put your Volo admin web ui password here
        )
    )
);
?>
```

To get output like this:

```
Array
(
    [212.220.145.10] => mx1.wowspanky.stream
    [212.220.145.100] => mx18.jokesclub.stream
    [212.220.145.101] => mx19.jokesclub.stream
    [212.220.145.102] => mx20.jokesclub.stream
    [212.220.145.103] => mx21.jokesclub.stream
    [212.220.145.104] => mx22.jokesclub.stream
    [212.220.145.105] => mx23.jokesclub.stream
    [212.220.145.106] => mx24.jokesclub.stream
    [212.220.145.107] => mx25.jokesclub.stream
    [212.220.145.108] => mx1.ohman.club
    [212.220.145.109] => mx2.ohman.club
    [212.220.145.11] => mx2.wowspanky.stream
    [212.220.145.110] => mx3.ohman.club
    [212.220.145.111] => mx4.ohman.club
    ...
)
```

So now you know the ips and the hostnames configured on this Volo box.

Let's send a drop!

Here's how:

```
<?php
require_once("utils.php");

$newMessageId = xu_rpc_http_concise(
    array(
        'method' => 'send_message',
        'args' => array(
            'user_id' => 4,
            'lists' => array('89'),
            'from' => 'Joe Pesci <jpesci@%$_sdomain%$', // ENDING POINTY BRACKET MISSING!!
            'subject' => 'Test by Joe Volo',
            'html' => '<html>Hello World </html>',
            'send_text' => 'no',
            'send_html' => 'yes',
            'send_aol' => 'no',
            'ip_space' => '*.jokesclub.stream',
        ),
        'host' => '<YOUR VOLO IP ADDRESS>', // put your Volo server ip here
        'uri' => '/xapi/server.php',
        'port' => 80,
        'user' => 'admin',
        'pass' => '<YOUR ADMIN UI PASSWORD>' // put your Volo admin web ui password here
    )
);
print_r($newMessageId);
?>
```

```
joejoe@mbp ~/SKRIPTTE/API$ php xcite.php
```

```
Array
(
    [faultCode] => 0
    [faultString] => 'from' is not valid
    [faultTrace] => #0 [internal function]:
xapi_send_message('send_message', Array, '')
#1 /home/vmp/htdocs/xapi/server.php(76):
xmlrpc_server_call_method(Resource id #27, '<?xml version="...", ''')
#2 {main}
)
```

Aha, so we get some debugging in there, nice. OK, let's add the missing '>' and try again:

```

<?php
require_once("utils.php");

$newMessageId = xu_rpc_http_concise(
    array(
        'method' => 'send_message',
        'args' => array(
            'user_id' => 4,
            'lists' => array('89'),
            'from' => 'Joe Pesci <jpesci@%$_sdomain%>', // WE ADDED THE MISSING '>'
            'subject' => 'Test by Joe Volo',
            'html' => '<html>Hello World </html>',
            'send_text' => 'no',
            'send_html' => 'yes',
            'send_aol' => 'no',
            'ip_space' => '*.jokesclub.stream',
        ),
        'host' => '<YOUR VOLO IP ADDRESS>', // put your Volo server ip here
        'uri' => '/xapi/server.php',
        'port' => 80,
        'user' => 'admin',
        'pass' => '<YOUR ADMIN UI PASSWORD>' // put your Volo admin web ui password here
    )
);
print_r($newMessageId);
?>

```

```

joejoe@mbp ~/SKRIPTTE/API$ php xcite.php
Array
(
    [0] => 603846
)

```

So we got the message id of the newly created drop, and now we can every once in a while pull logs for that drop, message id 603846.

Delivery Queue

Here's how to query the delivery queue:

```
<?php
require_once("utils.php");

print_r(
    xu_rpc_http_concise(
        array(
            'method' => 'delivery_queue',
            'args' => array(
                'user_id' => 4,
            ),
            'host' => '<YOUR VOLO IP ADDRESS>', // put your Volo server ip here
            'uri' => '/xapi/server.php',
            'port' => 80,
            'user' => 'admin',
            'pass' => '<YOUR ADMIN UI PASSWORD>' // put your Volo admin web ui password here
        )
    )
);
?>
```

And you'll get an array for every type of drop (paused, delivering, preparing, in queue, scheduled, finished, canceled, errored out):

```
Array
(
    [paused] => Array
        (
        )

    [delivering] => Array
        (
        )

    [preparing] => Array
        (
        )

    [in_queue_for_processing] => Array
        (
        )
)
```

```

[scheduled] => Array
(
)

[finished] => Array
(
    [data] => Array
        (
            [0] => Array
                (
                    [message_id] => 603846
                    [owner] => 4
                    [start_time] => 02/19/2018 20:00
                    [end_time] => 02/19/2018 20:00
                    [list_or_campaign_description] =>
Volotest

                    [list] => (subscribed(Volotest)) and
(field(bcount)<3)

                    [lists] => Volotest
                    [campaign] =>
                    [subject] => Test by Joe Volo
                    [tracking] =>
                    [tracking_indicator] => 0
                    [embedded_indicator] => 0
                    [autoresponder_indicator] => 0
                    [queued] => 1
                    [processed_percent] => 100.00
                    [processed] => 1
                    [delivered_percent] => 100.00
                    [delivered] => 1
...

```


message_logs method

So let's find out how our drop 603846 did:

```
<?php
require_once("utils.php");

print_r(
    xu_rpc_http_concise(
        array(
            'method' => 'message_logs',
            'args' => array(
                'message_id' => 603846
            ),
            'host' => '<YOUR VOLO IP ADDRESS>', // put your Volo server ip here
            'uri' => '/xapi/server.php',
            'port' => 80,
            'user' => 'admin',
            'pass' => '<YOUR ADMIN UI PASSWORD>' // put your Volo admin web ui password here
        )
    )
);
?>
```

The output:

```
Array
(
    [delivery_analysis] => Array
        (
            [[0] Delivered] => Array
                (
                    [download_link] =>
http://74.114.151.200:80/external/get_log.php?ticket=974ab1f9559ee4f8
22d220b84248d162
                )
            )

    [open_link] => Array
        (
            [download_link] =>
http://74.114.151.200:80/external/get_log.php?ticket=d8ea31c23a1ed957
6cb490e1583112da
        )
)
```

```
[track_links] => Array
  (
  )

[unsub_link] => Array
  (
  )

[complaints_link] => Array
  (
  )

)
```

So whatever the targets said during delivery, whatever bounce reply they issued - there's a separate file for each thing they said, and you got links to download (and then parse) those logs.

Volo + external DB

Normally a Volo box contains a MySQL database in which emails are organized into mailing lists, but what if you use multiple platforms? You have to keep the same dataset on all of them, because you do not want one platform to keep emailing people who already complained on another platform.

The solution is to keep data in a separate database and export some of that data when you are sending a drop. Once the drop is done, gather info on opens, clicks, bounces, unsubs and complaints and update your database. I am sure you know of one or two businesses that provide this feature for a fee.

So again: you export email addresses with accompanying data from your database into a file, upload that file to a Volo box (ftp, scp), and then use API to send an offer to emails listed in that file.

As an example, I exported some data from my database into a file called data.txt and placed that file somewhere on the Volo box, for instance at /home/biggie/import, here's a preview of the file:

```
[root@import]# cat data.txt
email,fname,lname,some_id
joe@volomp.com,Josephe,Beppo,123
moe@volomp.com,Moses,Brown,4463
...
```

I have extra columns in the file (fname, lname, some_id) whose names are unrelated to default Volo tags, yet you can use them freely.

The default delimiter is semi-colon (this: ';'), but you can override that with the 'extraseparator' option. No whitespaces are allowed around separators.

Method *send_message* returns message id, which you then use in the *message_logs* call to get download links for that drop (delivered, opens, tracks, unsubs, complaints)

Here's the script:

```
<?php
require_once("utils.php");

$newMessageId = xu_rpc_http_concise(
    array(
        'method' => 'send_message',
        'args' => array(
            'user_id' => 4,
            // 'lists' => array('89'),
            // no lists from the DB needed, we have a file
            'noquery' => '1',
            'extraseparator' => ',', // SEMI-COLON BY DEFAULT, USUALLY REPLACED LIKE THIS
            'extrafile' => '/home/biggie/import/data.txt', // FULL PATH TO FILE
            'from' => 'Joe Pesci <jpesci@%$_sdomain%>',
            'subject' => 'Test by Joe Volo',
            'html' => '<html>Hello World <br> %$_email% <br> %$_fname%, %$_lname%, %$_some_id%
<hr></html>',
            'send_text' => 'no',
            'send_html' => 'yes',
            'send_aol' => 'no',
            'ip_space' => '*.jokesclub.stream',
        ),
        'host' => '<YOUR VOLO IP ADDRESS>', // put your Volo server ip here
        'uri' => '/xapi/server.php',
        'port' => 80,
        'user' => 'admin',
        'pass' => '<YOUR ADMIN UI PASSWORD>' // put your Volo admin web ui password here
    )
);
print_r($newMessageId);
?>
```

So there you go, a really easy way to select some records from your DB, place it in a file on your Volo server and then send to people listed in the file, all the while using field names that make sense to you personally (as opposed to Volo default tags). And you get to use Volo throttling and delivery overview in the process.

Injection drop - DynQueue

Volo is designed from the start as a bulk mailer, our own MTA was designed as such, which means every drop is defined by two things:

- A fixed set of email addresses you are going to hit
- A content “template” comprised of:
 - [multiple] from fields
 - [multiple] subject lines
 - Reply-to field
 - Message bodies (html, txt, there used to be an AOL version as well)

There are Volo tags/macros (like `%%$first_name%%`) that you can place anywhere so just before the actual delivery those tags get replaced with the actual values from the database (or the operating system provides datetime, random strings, etc., depending on what the tag calls for).

So that’s pretty flexible, lots of Volo tags are available (see Message / New Message page), even more automatically appear if you add new columns to your database, and you can put them anywhere. The inflexible part is the set of emails. Whatever emails you marked for delivery - that’s it, no way to change that once a drop is created.

That’s where the DynQueue feature steps in - it lets you inject individual email addresses into already going drops, and that’s not all, because apart from using default volo tags you can also proclaim custom tags! In fact, you can proclaim complex custom tags (for instance the entire from field or the entire subject field or the entire message body) which contain default volo tags. With all three major parts of an email ready to be populated for each injection - you get unique, individualized emails for each recipient.

Some people think this is a Volo way to do transactional emails, or mimic a smtp relay. It may look like it a little bit, but we were not aiming for it, and is Volo still a bulk, templating engine, therefore Volo bunches up a couple of those injections into a small batch and sends it off to the engine. So there’s some lag between the first injection and the first attempted delivery.

Creating a DynQueue drop

You need to pass options noquery and dynqueue and also expandfields (to instruct volo to parse custom tags and look for default tags inside them), and you also have to list all the track link landing pages:

```
<?php
require_once("utils.php");
$result = xu_rpc_http_concise(
    array(
        'method' => 'send_message',
        'args' => array (
            'user_id'=> 11,
            'noquery' => 1, // NEEDED
            'dynqueue' => 1, // NEEDED
            'from'=>'%%$custom_from%',
            'subject'=>'%%$custom_subject%',
            'html'=>'%%$custom_body%',
            'send_text'=>'no',
            'send_html'=>'yes',
            'send_aol'=>'no',
            'expandfields' => array('custom_from', 'custom_subject', 'custom_body'),//NEEDED
            'ip_space' => '*.whatever.com',
            'tracked_links' => array( // YOU HAVE TO LIST ALL THE TRACK LINKS LANDING PAGES
                array
                (
                    "type" => "track",
                    "domain" => "http://www.landingpage1.com?e=%%$email%&sid=32",
                    "action_id" => 1
                ),
                array
                (
                    "type" => "track",
                    "domain" => "http://www.landingpage2.com?cid=44&bid=31",
                    "action_id" => 3
                )
            )
        ),
        'host' => '<YOUR VOLO IP ADDRESS>', // put your Volo server ip here
        'uri' => '/xapi/server.php',
        'port' => 80,
        // 'debug' => 2,
        'user' => 'admin',
        'pass' => '<YOUR ADMIN UI PASSWORD>' // put your Volo admin web ui password here
    )
);
print_r($result); // THIS IS THE MESSAGE_ID YOU WILL LATER ON USE TO INJECT INTO THIS DROP
?>
```

Injecting emails into a DynQueue drop

Now that the dynqueue drop is in there, permanently running, you can insert individual emails one by one, issuing calls like this. As you can see, you get to fill-out whatever “expandfields” (complex custom tags) you allowed for earlier.

```
<?php
require_once("utils.php");

$result = xu_rpc_http_concise(
    array(
        'method' => 'dynqueue',
        'args' => array (
            'message_id' => 21283880, // message id you got when you created the dynqueue
drop
            'info' => array(
                'email' => 'test@mailinator.com',
                'custom_from' => 'Joe King <joe@supermailer.com>'
                'custom_subject' => 'this is a unique subject for %$email% only',
                'custom_body' => '<html>
                    Hello %$first_name%,
                    <a href=""%$track.link1%">click here to win big!</a>
                    <br>
                    Another link: %$track.link2%
                    how are you today?...'
            )
        ),
        'host' => '<YOUR VOLO IP ADDRESS>', // put your Volo server ip here
        'uri' => '/xapi/server.php',
        'port' => 80,
        // 'debug' => 2,
        'user' => 'admin',
        'pass' => '<YOUR ADMIN UI PASSWORD>' // put your Volo admin web ui password here
    )
);
?>
```

Please note that you have to refer to the track links as %\$track.link<NUMBER>%\$.

Volo pools a couple of these injections and then sends the mini-batch to the engine queue. Dynqueue injections normally go out within a couple of minutes (in my tests usually much quicker, like 5-10 sec or so).

So with DynQueue you get to:

- Inject new recipients as the need arises
- Send customized (from/subject/body) emails
- The overhead compared to rolling out a new drop for each recipient is severely diminished (just think of all the accompanying data structures created with each drop)

The twist with the new DynQueue - Named DynQueue

Earlier in 2018 we added one small thing that completely changes the dynamic here. Namely, when creating a DynQueue drop we added the ability to put a name to that drop, like this:

```
<?php
require_once("utils.php");
$result = xu_rpc_http_concise(
    array(
        'method' => 'send_message',
        'args' => array (
            'user_id'=> 11,
            'noquery' => 1,
            //'dynqueue' => 1,
            'dynqueue' => 'FORGOTTEN_PASSWORD', // MAJOR DEVELOPMENT
            HERE!!!
            'from'=>'%%$custom_from%%',
            'subject'=>'%%$custom_subject%%',
            'html'=>'%%$custom_body%%',
            'send_text'=>'no',
            'send_html'=>'yes',
            ...
        )
    )
);
```

So for injecting instead of using the message_id you simply inject into a dynqueue drop called 'FORGOTTEN_PASSWORD', like this:

```
<?php
require_once("utils.php");

$result = xu_rpc_http_concise(
    array(
        'method' => 'dynqueue',
        'args' => array (
            'message_id' => 'FORGOTTEN_PASSWORD',
            'info' => array(
                'email' => 'test@mailinator.com',
                ...
            )
        )
    )
);
```


Do you see the implication? You can simply keep recreating that 'FORGOTTEN_PASSWORD' dynqueue drop whenever you want to change, say, the track link landing pages, and the injection script will simply continue working.

In fact, here's a script (by chance this one uses AmazonSES) that picks up any *.html files in the folder and makes/updates dynqueue drops that will be:

- Called like those html files
- Use content from those html files

```
#!/usr/bin/php
```

```
<?php
```

```
require_once 'utils.php';
```

```
$xapi_params = array(  
    'method' => 'send_message',  
    'args' => array(  
        'user_id' => '1',  
        'noquery' => '1',  
        'dynqueue' => 'PLACEHOLDER',  
  
        'from' => 'news@amazonapprovedfrom.net',  
        'subject' => 'Newsletter',  
        'html' => 'PLACEHOLDER',  
        'send_text' => 'no',  
        'send_html' => 'yes',  
        'send_aol' => 'no',  
        'aws' => array(  
            'AWSAccessKeyId' => 'AKIKAURISMAKI',  
            'AWSSecretKey' => 'hunter2',  
            'url' => 'email-smtp.eu-west-1.amazonaws.com'  
        )  
    ),
```

```
),  
  
    'host' => '<YOUR VOLO IP ADDRESS>', // put your Volo server ip here  
    'uri' => '/xapi/server.php',  
    'port' => 80,  
    'user' => 'admin',  
    'pass' => '<YOUR ADMIN UI PASSWORD>' // put your Volo admin web ui  
password here  
);
```

```
function update_template( $xp, $tplname, $contents )  
{
```

```

$xp['args']['html'] = $contents;
$xp['args']['dynqueue'] = $tplname;

$ret = xu_rpc_http_concise( $xp );

echo 'Updated ', $tplname, ' ( ', PHP_EOL;
var_dump( $ret );
}

foreach( glob('*.html') as $entry ) {
    $template = file_get_contents($entry);
    $tplsum = md5($template);

    $md5sum = @file_get_contents($entry . '.md5');

    if($md5sum != $tplsum) {
        $tplname = substr( $entry, 0, -5 );
        update_template( $xapi_params, $tplname, $template );
        file_put_contents($entry . '.md5', $tplsum);
    }
}
}

```

So what's the practical value here? Well, imagine that you have 'password forgotten' emails in 50 different languages, and that designers change the looks of those templates every once in a while. All your html designer has to do is update html files and plop them in this folder, and your [cron] script will update the dynqueue drops to reflect that change. All the while injector scripts just work, no need to update the message_id they inject into.

And then you have 50 other templates for 'hey, long time no see', 50 other for 'thank you for signing up' etc etc. Whenever your designers change the looks and the update script runs - any injection after that will carry the new looks.

If we did not add the named dynqueue drops you would have to juggle the message_ids, it's so much more pain it would be hardly doable at all.

Utils.php

Here's the content of the utils.php file known to work, in case you need it:

```
<?php

ini_set('memory_limit', '-1');

/*
  This file is part of, or distributed with, libXMLRPC - a C library for
  xml-encoded function calls.

  Author: Dan Libby (dan@libby.com)
  Epinions.com may be contacted at feedback@epinions-inc.com
*/

/*
  Copyright 2001 Epinions, Inc.

  Subject to the following 3 conditions, Epinions, Inc. permits you, free
  of charge, to (a) use, copy, distribute, modify, perform and display this
  software and associated documentation files (the "Software"), and (b)
  permit others to whom the Software is furnished to do so as well.

  1) The above copyright notice and this permission notice shall be included
  without modification in all copies or substantial portions of the
  Software.

  2) THE SOFTWARE IS PROVIDED "AS IS", WITHOUT ANY WARRANTY OR CONDITION OF
  ANY KIND, EXPRESS, IMPLIED OR STATUTORY, INCLUDING WITHOUT LIMITATION ANY
  IMPLIED WARRANTIES OF ACCURACY, MERCHANTABILITY, FITNESS FOR A PARTICULAR
  PURPOSE OR NONINFRINGEMENT.

  3) IN NO EVENT SHALL EPINIONS, INC. BE LIABLE FOR ANY DIRECT, INDIRECT,
  SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES OR LOST PROFITS ARISING OUT
  OF OR IN CONNECTION WITH THE SOFTWARE (HOWEVER ARISING, INCLUDING
  NEGLIGENCE), EVEN IF EPINIONS, INC. IS AWARE OF THE POSSIBILITY OF SUCH
  DAMAGES.

*/

/* xmlrpc utilities (xu)
 * author: Dan Libby (dan@libby.com)
 */

// ensure extension is loaded.
```

```

xu_load_extension();

// a function to ensure the xmlrpc extension is loaded.
// xmlrpc_epi_dir = directory where libxmlrpc.so.0 is located
// xmlrpc_php_dir = directory where xmlrpc-epi-php.so is located
function xu_load_extension($xmlrpc_php_dir="") {
    if(!extension_loaded('xmlrpc')) {
        $bSuccess = true;
        putenv("LD_LIBRARY_PATH=/usr/lib/php4/apache/xmlrpc/");
        if ($xmlrpc_php_dir) {
            $xmlrpc_php_dir .= '/';
        }
        if (!extension_loaded("xmlrpc")) {
            $bSuccess = dl($xmlrpc_php_dir . "xmlrpc-epi-php.so");
        }
    }
    return $bSuccess;
}

/* generic function to call an http server with post method */
function xu_query_http_post($request, $host, $uri, $port, $debug,
                            $timeout, $user, $pass, $secure=false) {
    $response_buf = "";
    if ($host && $uri && $port) {
        $content_len = strlen($request);

        $fsockopen = $secure ? "fsockopen_ssl" : "fsockopen";

        dbg1("opening socket to host: $host, port: $port, uri: $uri", $debug);
        $query_fd = $fsockopen($host, $port, $errno, $errstr, 10);

        if ($query_fd) {

            $auth = "";
            if ($user) {
                $auth = "Authorization: Basic " .
                    base64_encode($user . ":" . $pass) . "\r\n";
            }

            $http_request =
                "POST $uri HTTP/1.0\r\n" .
                "User-Agent: xmlrpc-epi-php/0.2 (PHP)\r\n" .
                "Host: $host:$port\r\n" .
                $auth .
                "Content-Type: text/xml\r\n" .
                "Content-Length: $content_len\r\n" .
                "\r\n" .
                $request;

            dbg1("sending http request:</h3> <xmp>\n$http_request\n</xmp>", $debug);

```

```

fputs($query_fd, $http_request, strlen($http_request));

dbg1("receiving response...", $debug);

$header_parsed = false;

$line = fgets($query_fd, 4096);
while ($line) {
    if (!$header_parsed) {
        if ($line === "\r\n" || $line === "\n") {
            $header_parsed = 1;
        }
        dbg2("got header - $line", $debug);
    }
    else {
        $response_buf .= $line;
    }
    $line = fgets($query_fd, 4096);
}

fclose($query_fd);
}
else {
    dbg1("socket open failed", $debug);
}
}
else {
    dbg1("missing param(s)", $debug);
}

dbg1("got response:</h3>. <xmp>\n$response_buf\n</xmp>\n", $debug);

return $response_buf;
}

function xu_fault_code($code, $string) {
    return array('faultCode' => $code,
        'faultString' => $string);
}

function find_and_decode_xml($buf, $debug) {
    if (strlen($buf)) {
        $xml_begin = substr($buf, strpos($buf, "<?xml"));
        if (strlen($xml_begin)) {

            $retval = xmlrpc_decode($xml_begin);
        }
        else {

```

```

        dbg1("xml start token not found", $debug);
    }
}
else {
    dbg1("no data", $debug);
}
return $retval;
}

```

```

/**
 * @param params    a struct containing 3 or more of these key/val pairs:
 * @param host      remote host (required)
 * @param uri       remote uri    (required)
 * @param port      remote port (required)
 * @param method    name of method to call
 * @param args      arguments to send (parameters to remote xmlrpc server)
 * @param debug     debug level (0 none, 1, some, 2 more)
 * @param timeout   timeout in secs. (0 = never)
 * @param user      user name for authentication.
 * @param pass      password for authentication
 * @param secure    secure. wether to use fsockopen_ssl. (requires special php build).
 * @param output    array. xml output options. can be null. details below:
 *
 * output_type: return data as either php native data types or xml
 * encoded. ifphp is used, then the other values are ignored. default = xml
 * verbosity:    determine compactness of generated xml. options are
 * no_white_space, newlines_only, and pretty. default = pretty
 * escaping:     determine how/whether to escape certain characters. 1 or
 * more values are allowed. If multiple, they need to be specified as
 * a sub-array. options are: cdata, non-ascii, non-print, and
 * markup. default = non-ascii | non-print | markup
 * version:     version of xml vocabulary to use. currently, three are
 * supported: xmlrpc, soap 1.1, and simple. The keyword auto is also
 * recognized to mean respond in whichever version the request came
 * in. default = auto (when applicable), xmlrpc
 * encoding:    the encoding that the data is in. Since PHP defaults to
 * iso-8859-1 you will usually want to use that. Change it if you know
 * what you are doing. default=iso-8859-1
 *
 * example usage
 *
 * $output_options = array('output_type' => 'xml',
 *                          'verbosity' => 'pretty',
 *                          'escaping' => array('markup', 'non-ascii',
 * 'non-print'),
 *                          'version' => 'xmlrpc',
 *                          'encoding' => 'utf-8'
 * );
 *
 * or

```

```

*
*           $output_options = array('output_type' => 'php');
*/
function xu_rpc_http_concise($params) {
    $host = $uri = $port = $method = $args = $debug = null;
    $timeout = $user = $pass = $secure = $debug = null;

    extract($params);

    // default values
    if(!$port) {
        $port = 80;
    }
    if(!$uri) {
        $uri = '/';
    }
    if(!$output) {
        $output = array(version => 'xmlrpc');
    }

    $response_buf = "";
    if ($host && $uri && $port) {
        $request_xml = xmlrpc_encode_request($method, $args, $output);
        $response_buf = xu_query_http_post($request_xml, $host, $uri, $port, $debug,
                                           $timeout, $user, $pass, $secure);

        $retval = find_and_decode_xml($response_buf, $debug);
    }
    return $retval;
}

/* call an xmlrpc method on a remote http server. legacy support. */
function xu_rpc_http($method, $args, $host, $uri="/", $port=80, $debug=false,
                    $timeout=0, $user=false, $pass=false, $secure=false) {
    return xu_rpc_http_concise(
        array(
            method => $method,
            args   => $args,
            host   => $host,
            uri    => $uri,
            port   => $port,
            debug  => $debug,
            timeout => $timeout,
            user   => $user,
            pass   => $pass,
            secure => $secure
        ));
}

```

```

function xu_is_fault($arg) {
    // xmlrpc extension finally supports this.
    return is_array($arg) ? xmlrpc_is_fault($arg) : false;
}

/* sets some http headers and prints xml */
function xu_server_send_http_response($xml) {
    header("Content-type: text/xml");
    header("Content-length: " . strlen($xml) );
    echo $xml;
}

function dbg($msg) {
    echo "<h3>$msg</h3>"; flush();
}
function dbg1($msg, $debug_level) {
    if ($debug_level >= 1) {
        dbg($msg);
    }
}
function dbg2($msg, $debug_level) {
    if ($debug_level >= 2) {
        dbg($msg);
    }
}

?>

```