How Do I...

How Do I... (Use Cases)

If you want to get a jumpstart on common API use cases, you came to the right place! Expand the text areas below for walkthroughs and code samples of API calls...

- How Do I... (Use Cases)
    - IP Blocks - Update Fields
    - IP Blocks - Assign / Subassign
    - DNS
    - DHCP
  - Python SDK:
    - IP Blocks

IP Blocks - Update Fields

Context: How do I update the notes field of an IP block using the API in PHP?

1) Start with providing instance information, API key, Secret Key, and DNS Server IP; set up the connection

```
<?php
//
// This file walks through an example of how to look up a block id number
// in ProVision, and then use it to attach a notes field
//
// supply the URL of your ProVision instance, your API key and your Secret key.
$proVisionURL = "https://ops.6connect.com/qa-4.2.2";
$apiKey = "32-5DAYTJEE2TZHOFOB";
$apiSecretKey = "48b278ec873bda473a323dbc467f8669";
// this example uses 6connect's PHP APIClient
require_once("APIClient.php");
// set up the connection
$apiClient = new APIClient($proVisionURL, $apiKey, $apiSecretKey);
```

2) Split the metadata you want to have showing in the notes, and find the block with which it should associate

```
// lets imagine we have some metadata in the following format:
//
$string = "10.1.245.5||DFW7|HP a5820x|its-erp.dfw7.us.corp||";
//
// And we want to insert the Colo, Server type, and hostname into the Notes field of the IP block

// first we split everything up
$pieces = explode("|", $string);
$ip = $pieces[0];
$colo = $pieces[2];
$type = $pieces[3];
$host = $pieces[4];

// then we pull the IP block using the API.
$params = array();
$params['block'] = "$ip/32";                                // the IP block we're looking for, with netmask
// make the call to the IPAM-GET endpoint
$apiResponse = $apiClient->sendRequest('ipam', 'get', $params);
if ($apiResponse->status != 1) {
        echo "Could not pull information for block: $ip/32 !\n";
        die();
}
if (trim($apiResponse->message) == "No blocks found.") {
        echo "IP block $ip/32 not found in ProVison!\n";
    die();
}

// we now have the ipObject associated with this IP block.  Lets get its block id.
$blockId = $apiResponse->data[0]['id'];
echo "IP block id: $blockId \n";
```

3) Update the block with the notes

```
// it is time to update the block with the new notes.
$notes = "$colo,$type,$host";
$params = array();
$params['id'] = $blockId;
$params['notes'] = $notes;
// make the call to the IPAM-UPDATE endpoint
$apiResponse = $apiClient->sendRequest('ipam', 'update', $params);

// and done!
echo $apiResponse->message . "\n";
```

IP Blocks - Assign / Subassign

Context: I unassigned an IP address and now it's in the Holding Tank. Now I want to assign an IP from the Holding Tank. I don't want to unassign an IP randomly, in case it is allocated to a Resource. What are my options?

There are 3 options:

1) If you know the specific IP, you can use use the ipam-get api call to determine if it is in Holding:

```
/api/v1/api.php?target=ipam&action=get&cidr=1.2.3.4/32

{
  id:1234,
  cidr:"1.2.3.4",
  ...
  resource_name:"Holding"
}
```

2) If you want to show all blocks/IPs in Holding, you can use the following ipam-get API call:

```
/api/v1/api.php?target=ipam&action=get&resourceQuery={"name":"Holding"}
```

3) If you know the block is in Holding, you can issue another ipam-unassign API call to move it from Holding to Available:

```
/api/v1/api.php?target=ipam&action=unassign&block=1.2.3.4/32
```

Context: I need to create a Resource Holder, assign them an IP block, then subassign some IPs out of that block to two new Resource Holders. What does this look like in Python?

We broke this up in a few steps so it's easier to link together.

1) Let's create a Resource Holder called "Ned"

```
query_string    = 'target=resource&action=add&meta[type]=entry&meta[section]=resource-holder&meta[name]=Ned'
query_string    += '&apiKey=' + api_key
hash            = base64.b64encode( hmac.new(api_secret_key, query_string, hashlib.sha256).digest() )
url               = base_url + '?' + query_string + '&hash=' + hash
print 'Create Ned resource holder'
print url, "\n"
data = json.load(urllib2.urlopen(url))
ned_resource_id = data['data']['id']
```

2) Now let's add the 213.29.27.0/24 IP block

```
query_string =    'target=ipam&action=add&rir=RIPE&block=213.29.27.0/24'
query_string    += '&apiKey=' + api_key
hash            = base64.b64encode( hmac.new(api_secret_key, query_string, hashlib.sha256).digest() )
url               = base_url + '?' + query_string + '&hash=' + hash
print 'Create 213.29.27.0/24 block'
print url, "\n"
data = json.load(urllib2.urlopen(url))
```

3) With the block in the system, we can assign 213.29.27.0/24 to "Ned" the Resource Holder

```
query_string =    "target=ipam&action=directAssign&block=213.29.27.0/24&resourceId=%d" % (ned_resource_id)
query_string    += '&apiKey=' + api_key
hash            = base64.b64encode( hmac.new(api_secret_key, query_string, hashlib.sha256).digest() )
url               = base_url + '?' + query_string + '&hash=' + hash
print 'Assign 213.29.27.0/24 block to Ned'
print url, "\n"
data = json.load(urllib2.urlopen(url))
```

4) Since we plan on assigning IPs out of this block, we should enable subassignments for 213.29.27.0/24

```
query_string =    'target=ipam&action=update&block=213.29.27.0/24&allowSubAssignments=true'
query_string    += '&apiKey=' + api_key
hash            = base64.b64encode( hmac.new(api_secret_key, query_string, hashlib.sha256).digest() )
url               = base_url + '?' + query_string + '&hash=' + hash
print 'Update 213.29.27.0/24 to allow sub assignments'
print url, "\n"
data = json.load(urllib2.urlopen(url))
```

5) Now let's create a Resource Holder "Tara"

```
query_string =    "target=resource&action=add&meta[type]=entry&meta[section]=resource-holder&meta[name]
=Tara&meta[parent_id]=%d" % (ned_resource_id)
query_string    += '&apiKey=' + api_key
hash            = base64.b64encode( hmac.new(api_secret_key, query_string, hashlib.sha256).digest() )
url               = base_url + '?' + query_string + '&hash=' + hash
print 'Create Tara resource holder'
print url, "\n"
data = json.load(urllib2.urlopen(url))
tara_resource_id = data['data']['id']
```

6) To keep it interesting, let's create another Resource Holder "Una"

```
query_string =    "target=resource&action=add&meta[type]=entry&meta[section]=resource-holder&meta[name]
=Una&meta[parent_id]=%d" % (ned_resource_id)
query_string    += '&apiKey=' + api_key
hash            = base64.b64encode( hmac.new(api_secret_key, query_string, hashlib.sha256).digest() )
url               = base_url + '?' + query_string + '&hash=' + hash
print 'Create Una resource holder'
print url, "\n"
data = json.load(urllib2.urlopen(url))
una_resource_id = data['data']['id']
```

7) Assign a /28 block from Ned's 213.29.27.0/24 to Tara

```
query_string =    "target=ipam&action=smartAssign&type=ipv4&rir=RIPE&mask=28&&resourceId=%
d&assignedResourceId=%d" % (tara_resource_id, ned_resource_id)
query_string    += '&apiKey=' + api_key
hash            = base64.b64encode( hmac.new(api_secret_key, query_string, hashlib.sha256).digest() )
url               = base_url + '?' + query_string + '&hash=' + hash
print 'Assign block from Ned\'s 213.29.27.0/24 to Tara'
print url, "\n"
data = json.load(urllib2.urlopen(url))
```

8) Then assign another /28 block from Ned's 213.29.27.0/24 to Una

```
query_string =    "target=ipam&action=smartAssign&type=ipv4&rir=RIPE&mask=28&&resourceId=%
d&assignedResourceId=%d" % (una_resource_id, ned_resource_id)
query_string    += '&apiKey=' + api_key
hash            = base64.b64encode( hmac.new(api_secret_key, query_string, hashlib.sha256).digest() )
url               = base_url + '?' + query_string + '&hash=' + hash
print 'Assign block from Ned\'s 213.29.27.0/24 to Una'
print url, "\n"
data = json.load(urllib2.urlopen(url))
```

DNS

Context: I need to set up a DNS server using ProVision's API in PHP, create a zone with a few simple records, and push it to the server.

1) Start with providing instance information, API key, Secret Key, and DNS Server IP

```
<?php
//
//
// supply the URL of your ProVision instance, your API key and your Secret key.
$proVisionURL = "https://ops.6connect.com/qa-4.2.2";
$apiKey = "Nnvz8xKZDQUWke6gDxb";
$apiSecretKey = "2YojRbrHnToPZ7cDeFBzcTAvcfMbPVmX";
// this example uses 6connect's PHP APIClient
require_once("APIClient.php");
// set up the connection
$apiClient = new APIClient($proVisionURL, $apiKey, $apiSecretKey);

// save this.  IP of the DNS Server we're creating.
$serverIp = "208.39.106.184";
```

2) Add a DNS server

```
// begin making api calls.  We begin by adding a simple DNS server.
$params = array();
$params['displayName'] = "Example Server";                                                    // the
pretty name of the DNS server
$params['server'] = "208.39.106.184";                                                           // the
IP of the DNS Server
$params['active'] =
1;                                                                                            // whether
or not this server is currently enabled
$params['transferType'] = "SCP";                                          // we are using an ISC Bind server which
we will communicate with via SCP
$params['username'] = "6connect";                                                                //
the username used to SCP zones to this server
$params['password'] = "password";                                                                //
the password used to SCP zones to this server
$params['port'] = 22;                                                     // the port used to SCP zones to this
server
$params['serverType'] = "master";                                                                //
whether this server is a master or a slave
$params['SOA'] = "ns1.dns.6connect.net. hostmaster.6connect.net.";        // the default SOA
$params['remoteDirectory'] = "/tmp/";                                                           //
where to place the zone files on the server
$params['namedConfPath'] = "/tmp/";
// the path to the zones within the configuration file.  Usually the same as 'remoteDirectory'
$params['postCommand'] = "touch /tmp/allFinished";                                          // the command to
execute on the server after the transfer is complete.
// add the server
$apiResponse = $apiClient->sendRequest('dnsServer', 'add', $params);
if ($apiResponse->status == 1) {
        echo "Successfully added DNS Server '" . $params['displayName'] . "'\n";
} else {
        echo "Could not add DNS Server '" . $params['displayName'] . "' !\n";
        die();
}

// now we fetch the id of our newly created server
$params = array();
$apiResponse = $apiClient->sendRequest('dnsServer', 'get', $params);
$data = $apiResponse->data;
for ($i = 0; $i < count($data); $i++) {
        if ($data[$i]['server'] == $serverIp) {
                        // we save the id for later.
                        $serverId = $data[$i]['id'];
                        break;
        }
}
echo "Server Id is: $serverId \n";
```

3) Create a zone

```
// okay, DNS server is set up -- time to create a zone.
$params = array();
$params['zoneName'] = "atestzone.com";                                 // zone name
$params['zoneResourceId'] = 1;                                                 // the owner of the zone; 1 is
default
$apiResponse = $apiClient->sendRequest('zone', 'add', $params);
if ($apiResponse->status == 1) {
    echo "Successfully added DNS Zone '" . $params['zoneName'] . "'\n";
} else {
    echo "Could not add DNS Zone '" . $params['zoneName'] . "' !\n";
    die();
}
// snag the zoneId for later.
$zoneId = $apiResponse->data;
```

4) Add Zone records

```
// Lets add some records to our new zone!
$params = array();
$params['newRecordZoneId'] = $zoneId;                           // parent zone id
$params['newRecordType'] = 'A';                                       // record type
$params['newRecordHost'] = "www";                     // the host field of the record
$params['newRecordValue'] = "1.2.3.4";                // the value field of the record
$params['newRecordTTL'] = "3600";                     // the value of the TTL field
$apiResponse = $apiClient->sendRequest('record', 'add', $params);
if ($apiResponse->status == 1) {
    echo "Successfully added Record to zone #$zoneId\n";
} else {
    echo "Could not add Record to zone #$zoneId!\n";
    die();
}

$params = array();
$params['newRecordZoneId'] = $zoneId;                  // parent zone id
$params['newRecordType'] = 'A';                        // record type
$params['newRecordHost'] = "dev";                      // the host field of the record
$params['newRecordValue'] = "2.3.4.5";                     // the value field of the record
$params['newRecordTTL'] = "3600";                                     // the value of the TTL field
$apiResponse = $apiClient->sendRequest('record', 'add', $params);
if ($apiResponse->status == 1) {
    echo "Successfully added Record to zone #$zoneId\n";
} else {
    echo "Could not add Record to zone #$zoneId!\n";
    die();
}

$params = array();
$params['newRecordZoneId'] = $zoneId;                  // parent zone id
$params['newRecordType'] = 'A';                        // record type
$params['newRecordHost'] = "cloud";                    // the host field of the record
$params['newRecordValue'] = "3.4.5.6";                 // the value field of the record
$params['newRecordTTL'] = "3600";                      // the value of the TTL field
$apiResponse = $apiClient->sendRequest('record', 'add', $params);
if ($apiResponse->status == 1) {
    echo "Successfully added Record to zone #$zoneId\n";
} else {
    echo "Could not add Record to zone #$zoneId!\n";
    die();
}
```

4) Link the Zone to the new DNS server and push

```
// Okay, we have some zones with records.  Time to link this zone to the new DNS Server
$params = array();
$params['serverId'] = $serverId;                                                  // the server id
$params['zoneId'] = $zoneId;                                                         // the zone id
$params['serverSlave'] = 0;                                                                 // not a slave
zone
$apiResponse = $apiClient->sendRequest('zoneLinkage', 'add', $params);
if ($apiResponse->status == 1) {
    echo "Successfully linked Zone #$zoneId to server #serverId\n";
} else {
    echo "Could not link Zone #$zoneId to server #serverId!\n";
    die();
}
// now we can push the zone to the server
$params = array();
$params['zoneId'] = $zoneId;                            // the zone id to push
$apiResponse = $apiClient->sendRequest('dnsServer', 'transferSingle', $params);
if ($apiResponse->status == 1) {
    echo "Zone pushed!\n";
} else {
    echo "Could not push zone!\n";
    die();
}
?>
```

DHCP

Context: I need to attach the DHCP module as a child

DHCPv2 functionality is enabled on a particular resource by attaching a DHCP Module as a child. A command to do this is as follows:

```
            [ProVision root]/api/v1/api.php?target=resource&action=add

            data:
meta[type]: dhcp_module
meta[name]: [parent resource id] DHCP Module
meta[parent_id]: [parent resource id]
```

The special resource type “dhcp_module” indicates to ProVision that the DHCP system is enabled for the parent object. The attributes associated with the “dhcp_module” resource govern the DHCP system's behavior.

Updating the attributes of a DHCP Server uses a Resource Update command:

```
[ProVision root]/api/v1/api.php?target=resource&action=update&meta[id]=2178 &meta[type]=dhcp_module&fields
[_dhcp_attributes][]={"type":"ISC","notes":"notes go here","username":"username","port":"port","
config_test":"/etc/init.d/dhcpd configtest","server_stop":"/etc/init.d/dhcpd stop","server_start":"/etc/init.
d/dhcpd start","config_path":"/tmp/dhcpd.conf","option_routers":"192.168.0.0","option_domain_name_servers":"
ns1.6connect.com","option_domain_name":"6connect.com","authoritative":"1","default_lease_time":"600","
max_lease_time":"7200","local_port":"67","log_facility":"local7","password":"password","server_ip":"
192.168.0.1","freeLines":3,"freeLine1":"free line 1","freeLine2":"free line 2","freeLine3":"free line 3"}
```

This command appears rather complicated, but can be broken apart into reasonable pieces. The first section:

```
target=resource&action=update&meta[id]=2178&meta[type]=dhcp_module
```

is familiar from other parts of ProVision. We are updating a resource of type “dhcp_module” whose resource id is 2178. The second section of the command details the update values, starting with

```
 fields[_dhcp_attributes][]=
```

which contains a JSON-encoded string of all the fields specific to a DHCP server's function. When expanded into its full object form it is substantially easier to digest:

```
{
            "type":"ISC",
            "notes":"notes go here",
            "username":"username",
            "port":"port",
            "config_test":"/etc/init.d/dhcpd configtest",
            "server_stop":"/etc/init.d/dhcpd stop",
            "server_start":"/etc/init.d/dhcpd start",
            "config_path":"/tmp/dhcpd.conf",
            "option_routers":"192.168.0.0",
            "option_domain_name_servers":"ns1.6connect.com",
            "option_domain_name":"6connect.com",
            "authoritative":"1",
            "default_lease_time":"600",
            "max_lease_time":"7200",
            "local_port":"67",
            "log_facility":"local7",
            "password":"password",
            "server_ip":"192.168.0.1",
            "freeLines":3,
            "freeLine1":"free line 1",
            "freeLine2":"free line 2",
            "freeLine3":"free line 3"
}
```

This object describes all the most common DHCP server configuration options. For a full explanation of each of the fields, see the Detailed API Specification later in this document.

Please note that the object above must be passed to the DHCP system as a JSON-encoded string. It must be passed into the special "_dhcp_attributes" attribute for it to be functional, as in the example URL.

Context: I need to add a DHCP aggregate

An example command to add a DHCP Aggregate is:

```
[ProVision root]/api/v1/api.php?target=ipam&action=add&block=192.168.0.0/24&rir=
1918&vlan=&tags=&region=&resourceId=1282&allowSubAssignments=true
```

The important part to note is that the IP block is being assigned to resourceId 1282, which corresponds to the DHCP Available resource. The DHCP Available resource is a system-level resource which is used to hold all unassigned DHCP IP addresses. Every instance has its own DHCP Available resource, whose id can be found with the following command:

```
[ProVision root]/api/v1/api.php?target=resource&action=get&slug=dhcp-available
```

New DHCP subnets and hosts draw their IPs from this pool. If there are no IPs in the DHCP Available pool new subnets and hosts will not be able to be created.

DHCP IP aggregates are fetched, updated, split, and deleted using the standard IPAM management API endpoints. Please see the IPAM API Documentation. for details.

Context: I need to add a DHCP Pool

Similar to how the "dhcp_module" resource was created above, the command to create a DHCP Pool is as follows:

```
[ProVision root]/api/v1/api.php?target=resource&action=add&meta[type]=dhcp_pool &meta[name]=New Subnet&fields
[_dhcp_type][]=subnet&fields[_dhcp_pool_attributes][]={"mac":"","rangeStart":"","rangeEnd":"","freeLines":3,"
freeLine1":"Free Line 1","freeLine2":"Free Line 2","freeLine3":"Free Line 3"}
```

The first half of this command is relatively straightforward:

```
target=resource&action=add&meta[type]=dhcp_pool&meta[name]=New Subnet
```

This section informs the API that we wish to create a new, empty “dhcp_pool” resource whose name is “New Subnet.”

```
fields[_dhcp_type][]=subnet&fields[_dhcp_pool_attributes][]={"mac":"","rangeStart":"", "rangeEnd":"","
freeLines":3,"freeLine1":"Free Line 1","freeLine2":"Free Line 2","freeLine3":"Free Line 3"}
```

The second half of the command behaves in a similar manner to the “dhcp_module.” The “_dhcp_pool_attributes” field holds a JSON-encoded string which describes the dhcp_pool resource. When expanded, the JSON string becomes the following object:

```
{
            "mac":"",
            "rangeStart":"",
            "rangeEnd":"",
            "freeLines":3,
            "freeLine1":"Free Line 1",
            "freeLine2":"Free Line 2",
            "freeLine3":"Free Line 3"
}
```

For a full explanation of each of the fields, see the Detailed API Specification.

⚠ Please note that the object above must be passed to the DHCP system as a JSON-encoded string. It must be passed into the “_dhcp_pool_attributes” attribute for it to be functional, as in the example URL.

Once a dhcp_pool resource is in the system it can be updated with IP data obtained from the IP Management system. Under DHCPv2, the DHCP system uses all the standard IPAM API endpoints and can make use of both the smartAssign and the directAssign methods. Please see the IPAM API documentation for details.

Context: I need to link a DHCP pool to a DHCP server

An example of building a link between a dhcp_pool and a DHCP Server is:

```
[ProVision root]/api/v1/api.php?target=resource&action=addLink&resource_id1=2178&
resource_id2=1452&relation=dhcpPoolLink
```

The Resource Linkage system controls which DHCP Pools are associated with a given DHCP Server. In the case of linking a DHCP Pool to a DHCP Server, the relation used is “dhcpPoolLink”. This is a directional link, so it is important that resource_id1 and resource_id2 do not get confused.

```
relation:  "dhcpPoolLink"
resource_id1:  the id of the dhcp_module this pool is being linked to
resource_id2:  the id of the dhcp_pool being linked
```

⚠ It is very important that resource_id1 not be confused with resource_id2. The link will not function with the values reversed.

To undo the above and break a DHCP Pool link, use the same command but substitute “deleteLink” for the action “addLink”.

```
[ProVision root]/api/v1/api.php?target=resource&action=deleteLink&resource_id1=2178&
resource_id2=2179&relation=dhcpPoolLink
```

Context: I need to push a DHCP config file

Once the server has been configured according to the previous sections, hitting the following API endpoint will trigger a DHCP push:

```
[ProVision root]/api/v1/api.php?target=dhcp&action=push&id=2178
```

The “id” in the above string is the id of the dhcp_module resource attached to the server you whose configuration is to be pushed. The API return payload will contain success or failure codes, as well as a description of any errors which might have occurred.

When a DHCP configuration file is pushed an SSH connection is opened to the configured server using the user, password, and port supplied to the '_dhcp_attributes' attribute on the dhcp_module resource. If the system successfully connects, it will assemble a DHCP configuration from the information given to the dhcp_module's '_dhcp_attribute' attribute and then parse and add in all linked dhcp_pool resources.

After the assembled file has been transferred to the DHCP server it will be placed in the location given by 'config_path' on the dhcp_module, and then the command described in 'config_test' will be run to determine whether or not this new file parses correctly. If 'config_test' is blank or omitted, this step is skipped.

If the file parses correctly the DHCP will be stopped and restarted according to the 'server_stop' and 'server_start' commands on the DHCP module. If there are errors at any point the system backs out, replaces old config files, and reports the errors via the 'message' return field of the API call.

Python SDK:

IP Blocks

Context: How do I create aggregates, get block information, and delete aggregates using the API / python SDK?

```
#!/usr/bin/python
from apiclient import APIClient, APIResponse

# REPLACE WITH CORRECT VALUES FOR YOUR INSTANCE

base_url                = 'https://<ProVision Instance URL>'

api_key                       = '00-ABCDEFGHIJ123456'

api_secret_key        = '0123456789abcdef0123456789abcdef'


# create the APIClient

client = APIClient(base_url, api_key, api_secret_key)


# create aggregate 1.2.3.0/24

target = 'ipam'

action = 'add'

params = {'block': '1.2.3.0/24', 'rir':'ARIN'}

url = client.get_request_url(target, action, params)

print url

response = client.make_api_call(target, action, params)

print response


# get block 1.2.3.0/24

target = 'ipam'

action = 'get'

params = {'block': '1.2.3.0/24'}

url = client.get_request_url(target, action, params)

print url

response = client.make_api_call(target, action, params)

print response



# delete aggregate 1.2.3.0/24

target = 'ipam'

action = 'delete'

params = {'block': '1.2.3.0/24'}

url = client.get_request_url(target, action, params)

print url

response = client.make_api_call(target, action, params)

print response
```