

In-App Purchase (IAP) allows providers of ongoing services through Odoo apps to be compensated for ongoing service use rather than — and possibly instead of — a sole initial purchase.

In that context, Odoo acts mostly as a *broker* between a client and an Odoo App Developer:

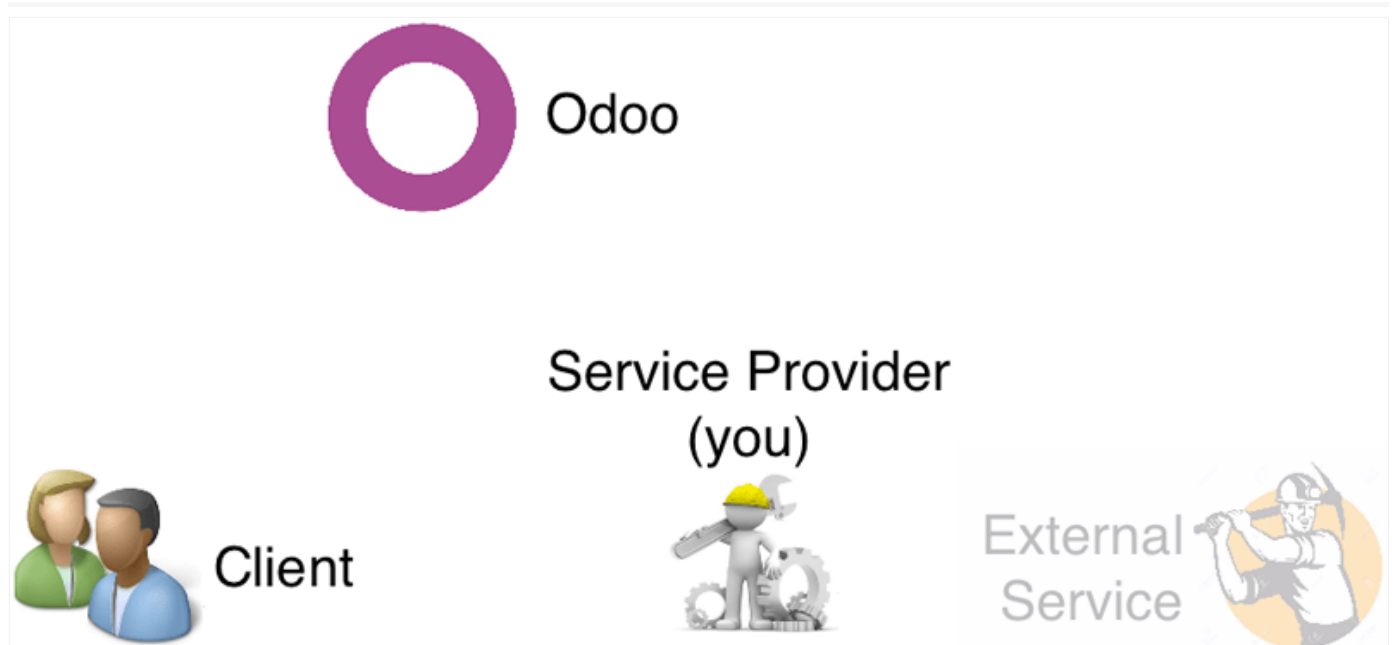
Users purchase service tokens from Odoo.

Service providers draw tokens from the user's Odoo account when service is requested.

### ▲ Attention

This document is intended for *service providers* and presents the latter, which can be done either via direct **JSON-RPC2** (<https://www.jsonrpc.org/specification>), or if you are using Odoo using the convenience helpers it provides.

## Overview



## The Players

The Service Provider is (probably) you the reader, you will be providing value to the client in the form of a service paid per-use.

The Client installed your Odoo App, and from there will request services.

Odoo brokers crediting, the Client adds credit to their account, and you can draw credits from there to provide services.

The External Service is an optional player: *you* can either provide a service directly, or you can delegate the actual service acting as a bridge/translator between an Odoo system and the actual service.



## The Credits

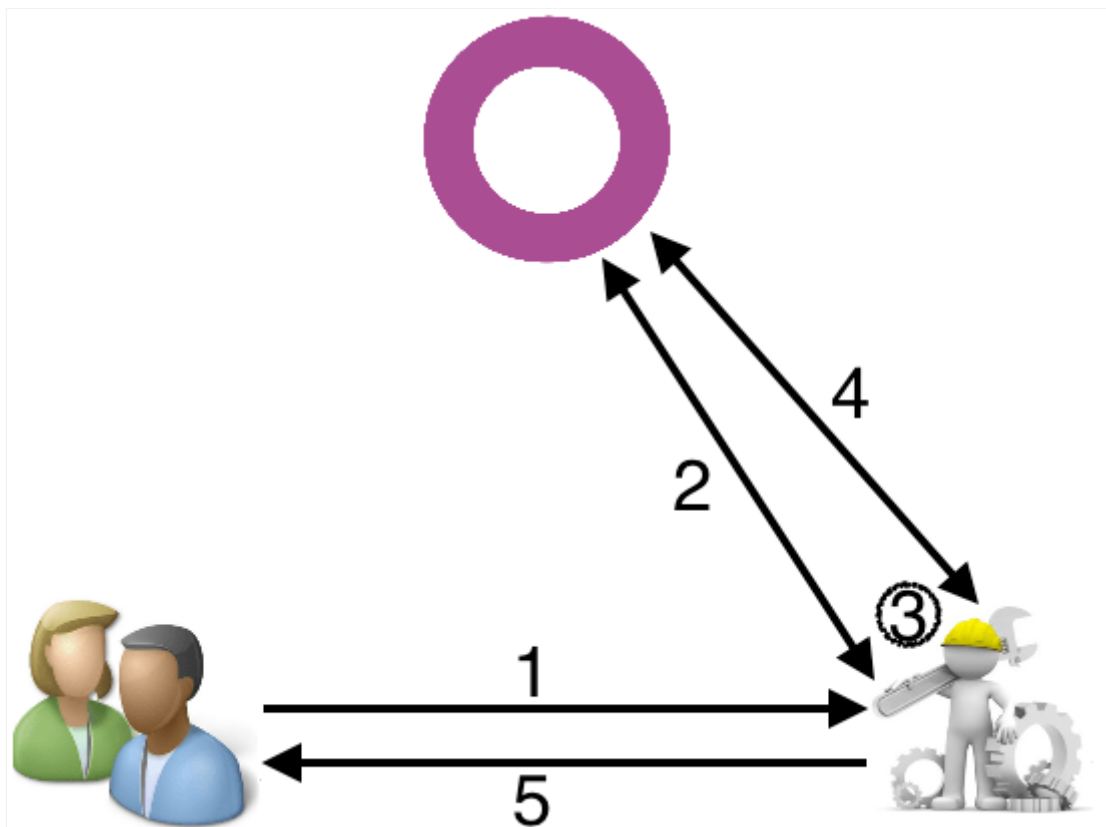
The credits went from integer to float value starting **October 2018**. Integer values are still supported. Every service provided through the IAP platform can be used by the clients with tokens or *credits*. The credits are an float unit and their monetary value depends on the service and is decided by the provider. This could be:

- for an sms service: 1 credit = 1 sms;
- for an ad service: 1 credit = 1 ad; or
- for a postage service: 1 credit = 1 post stamp.

A credit can also simply be associated with a fixed amount of money to palliate the variations of price (e.g. the prices of sms and stamps may vary following the countries).

The value of the credits is fixed with the help of prepaid credit packs that the clients can buy on <https://iap.odoo.com> (<https://iap.odoo.com>) (see **Packs**).

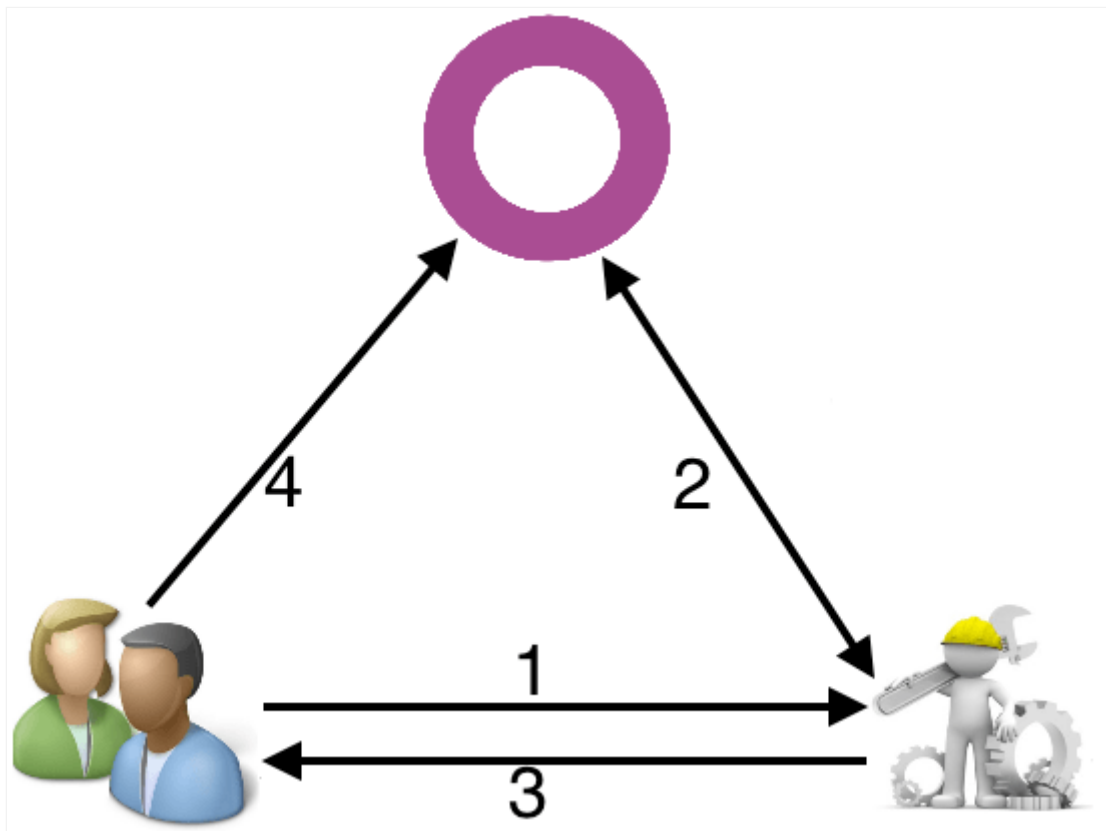
In the following explanations we will ignore the External Service, they are just a detail of the service you provide.



## 'Normal' service flow

If everything goes well, the normal flow is the following:

- 1 The Client requests a service of some sort.
- 2 The Service Provider asks Odoo if there are enough credits for the service in the Client's account, and creates a transaction over that amount.
- 3 The Service Provider provides the service (either on their own or calling to External Services).
- 4 The Service Provider goes back to Odoo to capture (if the service could be provided) or cancel (if the service could not be provided) the transaction created at step 2.
- 5 Finally, the Service Provider notifies the Client that the service has been rendered, possibly (depending on the service) displaying or storing its results in the client's system.



## Insufficient credits

However, if the Client's account lacks credits for the service, the flow will be as follows:

- 1 The Client requests a service as previously.
- 2 The Service Provider asks Odoo if there are enough credits on the Client's account and gets a negative reply.
- 3 This is signaled back to the Client.
- 4 Who is redirected to their Odoo account to credit it and re-try.

# Building your service

For this example, the service we will provide is ~~mining dogecoins~~ burning 10 seconds of CPU for a credit. For your own services, you could, for example:

- provide an online service yourself (e.g. convert quotations to faxes for business in Japan);
- provide an *offline* service yourself (e.g. provide accountancy service); or
- act as intermediary to an other service provider (e.g. bridge to an MMS gateway).

## Register the service on Odoo

The first step is to register your service on the IAP endpoint (production and/or test) before you can actually query user accounts. To create a service, go to your *Portal Account* on the IAP endpoint (<https://iap.odoo.com> (<https://iap.odoo.com>) for production, <https://iap-sandbox.odoo.com> (<https://iap-sandbox.odoo.com>) for testing, the endpoints are *independent* and *not synchronized*). Alternatively, you can go to your portal on Odoo (<https://iap.odoo.com/my/home> (<https://iap.odoo.com/my/home>)) and select *In-App Services*.

On production, there is a manual validation step before the service can be used to manage real transactions. This step is automatically passed when on sandbox to ease the tests.

Log in then go to **My Account ▶ Your In-App Services**, click Create and provide the informations of your service.

The service has seven important fields:

**name** - **ServiceName** : This is the string you will need to provide inside the client's `app` when requesting a transaction from Odoo. (e.g. `self.env['iap.account'].get(name)` ). As good practice, this should match the technical name of your app.

**label** - **Label** : The name displayed on the shopping portal for the client.

### ▲ Warning

Both the **ServiceName** and **Label** are unique. As good practice, the **ServiceName** should usually match the name of your Odoo Client App.

**icon** - **Icon** : A generic icon that will serve as default for your `packs`.

**key** - **ServiceKey** : The developer key that identifies you in IAP (see [your service](#)) and allows to draw credits from the client's account. It will be shown only once upon creation of the service and can be regenerated at will.

### ▲ Danger

Your **ServiceKey** *is a secret*, leaking your service key allows other application developers to draw credits bought for your service(s).

**trial credits** - **Float** : This corresponds to the credits you are ready to offer upon first use to your app users. Note that such service will only be available to clients that have an active enterprise contract.

**privacy policy - PrivacyPolicy** : This is an url to the privacy policy of your service. This should explicitly mention the **information you collect**, how you **use it, its relevance** to make your service work and inform the client on how they can **access, update or delete their personal information**.

## Your Documents

My Opportunities	0
Sales Orders	0
Invoices	0
Projects	0
Tickets	0
Support Packs	There are currently no support packs for your account.
In-App Services	

## Your Services

[Update Payment Info](#)

This is a production platform ! If you want to create a test service, please go to the [sandbox](#) and refer to the [documentation](#).

Name	Packs	Status
otherside	0	Active
reveal	2	Active
sms	1	Active


[Create](#)

### Edit service

#### Technical Name

This is the string you will need to provide in your module when requesting a transaction from Odoo IAP.  
(e.g. `self.env['iap.account'].get(service_name)`)

#### Service Logo



Suggested image size: 360x150px

[Choose file](#) No file chosen

#### Label

The name that will be displayed to users.

#### Description

A short description of the service you are offering.

#### Unit Name

If left blank, 'Credits' will be used as the unit of your service.

#### Trial Credits

Upon first use of your service, those trial credits will be granted for free to the customer (see [the documentation](#)).


#### Privacy Policy

You need to provide a URL with the privacy policy of your service. This policy should inform the user about several points. Amongst them:

- the information you collect
- how you use this information
- how the client can access his data
- how you protect this data
- ...

You can refer to the [Odoo Privacy Policy](#) as an example of structure but you can **NOT** use it as your own privacy policy.


[Cancel](#) [Save](#)



### Service key generated

Be sure to write it down as we will show it **only once** and **we do not store it**.

service key: `34d9b6612f424c46803c9dbfd5a86e5f`



## reveal

My description

Status Waiting for Approval

Total purchases 0.00 Queries

[Edit](#)

[Reset service key](#)

### Packs

[Create New](#)

There are currently no packs for this service.

You can then create *credit packs* which clients can purchase in order to use your service.

## Packs

A credit pack is essentially a product with five characteristics:

Name: name of the pack,

Icon: specific icon for the pack (if not provided, it will fallback on the service icon),

Description: details on the pack that will appear on the shop page as well as the invoice,

Amount: amount of credits the client is entitled to when buying the pack,

Price: price in EUR (for the time being, USD support is planned).

Odoo takes a 25% commission on all pack sales. Adjust your selling price accordingly.

Depending on the strategy, the price per credit may vary from one pack to another.

Edit pack

Pack Name

500 credits

Description

This will allow 500 calls the reveal service

Amount

500.0

Queries

Price

100.0

€

Note that Odoo will take a 25% commission on each sale. Define your price accordingly.

Pack Icon

There's no place like 127.0.0.1

Choose file

No file chosen

Cancel

Save

## Odoo App

The second step is to develop an Odoo App (<https://www.odoo.com/apps>) which clients can install in their Odoo instance and through which they can *request* the services you provide. Our app will just add a button to the Partners form which lets a user request burning some CPU time on the server.

First, we will create an *odoo module* depending on **iap**. IAP is a standard V11 module and the dependency ensures a local account is properly set up and we will have access to some necessary views and useful helpers.

*coalroller/\_manifest\_.py*

```
{
    'name': "Coal Roller",
    'category': 'Tools',
    'depends': ['iap'],
}
```

*coalroller/\_init\_.py*



```
# -*- coding: utf-8 -*-
```

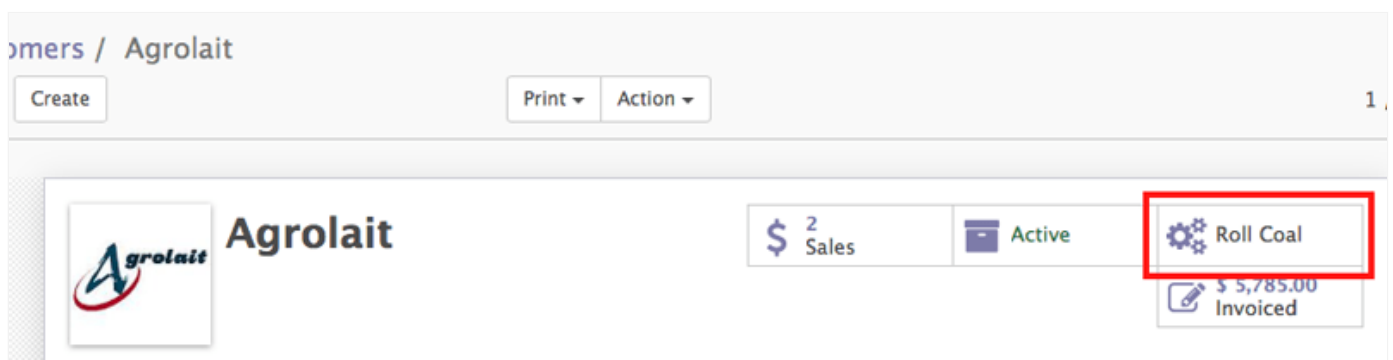
Second, the “local” side of the integration. Here we will only be adding an action button to the partners view, but you can of course provide significant local value via your application and additional parts via a remote service.

*coalroller/\_\_manifest\_\_.py*

```
{
    'name': "Coal Roller",
    'category': 'Tools',
    'depends': ['iap'],
    'data': [
        'views/views.xml',
    ],
}
```

*coalroller/views/views.xml*

```
<odoo>
<record model="ir.ui.view" id="partner_form_coalroll">
    <field name="name">partner.form.coalroll</field>
    <field name="model">res.partner</field>
    <field name="inherit_id" ref="base.view_partner_form" />
    <field name="arch" type="xml">
        <xpath expr="//div[@name='button_box']">
            <button type="object" name="action_partner_coalroll"
                    class="oe_stat_button" icon="fa-gears">
                <div class="o_form_field o_stat_info">
                    <span class="o_stat_text">Roll Coal</span>
                </div>
            </button>
        </xpath>
    </field>
</record>
</odoo>
```



We can now implement the action method/callback. This will *call our own server*.

There are no requirements when it comes to the server or the communication protocol between the app and our server, but `iap` provides a `iap_jsonrpc()` helper to call a JSON-RPC2 (<https://www.jsonrpc.org/specification>) endpoint on an other Odoo instance and transparently



re-raise relevant Odoo exceptions ( `InsufficientCreditError` , `odoo.exceptions.AccessError` (`../reference/orm.html#odoo.exceptions.AccessError`) and `odoo.exceptions.UserError` (`../reference/orm.html#odoo.exceptions.UserError`)).

In that call, we will need to provide:

- any relevant client parameter (none here),
- the `token` of the current client that is provided by the `iap.account` model's `account_token` field. You can retrieve the account for your service by calling `env['iap.account'].get(service_name)` where `service_name` is the name of the service registered on IAP endpoint.

*coalroller/\_\_init\_\_.py*

```
# -*- coding: utf-8 -*-
from . import models
```

*coalroller/models/\_\_init\_\_.py*

```
from . import res_partner
```

*coalroller/models/res\_partner.py*

```
# -*- coding: utf-8 -*-
from odoo import api, models
from odoo.addons.iap import jsonrpc, InsufficientCreditError

# whichever URL you deploy the service at, here we will run the remote
# service in a local Odoo bound to the port 8070
DEFAULT_ENDPOINT = 'http://localhost:8070'
class Partner(models.Model):
    _inherit = 'res.partner'
    def action_partner_coalroll(self):
        # fetch the user's token for our service
        user_token = self.env['iap.account'].get('coalroller')
        params = {
            # we don't have any parameter to provide
            'account_token': user_token.account_token
        }
        # ir.config_parameter allows locally overriding the endpoint
        # for testing & al
        endpoint = self.env['ir.config_parameter'].sudo().get_param('coalroller.endpoint')
        jsonrpc(endpoint + '/roll', params=params)
        return True
```

`iap` automatically handles `InsufficientCreditError` coming from the action and prompts the user to add credits to their account.

`iap_jsonrpc()` takes care of re-raising `InsufficientCreditError` for you.

### ▲ Danger

If you are not using `iap_jsonrpc()` you *must* be careful to re-raise `InsufficientCreditError` in your handler otherwise the user will not be prompted to credit their account, and the next call will fail the same way.

## Service

Though that is not *required*, since `iap` provides both a client helper for JSON-RPC2 (<https://www.jsonrpc.org/specification>) calls ( `iap_jsonrpc()` ) and a service helper for transactions ( `iap_charge` ) we will also be implementing the service side as an Odoo module:

`coalroller_service/__init__.py`

```
# -*- encoding: utf-8 -*-
```

`coalroller_service/__manifest__.py`

```
{
    'name': "Coal Roller Service",
    'category': 'Tools',
    'depends': ['iap'],
}
```

Since the query from the client comes as JSON-RPC2 (<https://www.jsonrpc.org/specification>) we will need the corresponding controller which can call `iap_charge` and perform the service within:

`coalroller_service/controllers/main.py`

```
import time
```

```
from passlib import pwd, hash
```

```
from odoo import http
```

```
from odoo.addons.iap import charge
```

```
class CoalBurnerController(http.Controller):
```

```
    @http.route('/roll', type='json', auth='none', csrf='false')
```

```
    def roll(self, account_token):
```

```
        # the service key is a secret*, it should not be committed in
```

```
        # the source
```

```
        service_key = self.env['ir.config_parameter'].sudo().get_param('coalroller.servi
```

```
        # we charge 1 credit for 10 seconds of CPU
```

```
        cost = 1
```

```
        # TODO: allow the user to specify how many (tens of seconds) of CPU they want to
        with charge(http.request.env, service_key, account_token, cost):
```

```
            # 10 seconds of CPU per credit
```

```
            end = time.time() + (10 * cost)
```

```
            while time.time() < end:
```

```
                # we will use CPU doing useful things: generating and
```

```
                # hashing passphrases
```

```
                p = pwd.genphrase()
```

```
                h = hash.pbkdf2_sha512.hash(p)
```

```
        # here we don't have anything useful to the client, an error
```

```
        # will be raised & transmitted in case of issue, if no error
```

```
        # is raised we did the job
```

`coalroller_service/controllers/__init__.py`

```
# -*- encoding: utf-8 -*-
from . import main
```

*coalroller\_service/\_\_init\_\_.py*

```
# -*- encoding: utf-8 -*-
from . import controllers
```

The `iap_charge` helper will:

- 1 authorize (create) a transaction with the specified number of credits, if the account does not have enough credits it will raise the relevant error
- 2 execute the body of the `with` statement
- 3 if the body of the `with` executes successfully, update the price of the transaction if needed
- 4 capture (confirm) the transaction
- 5 otherwise, if an error is raised from the body of the `with`, cancel the transaction (and release the hold on the credits)

### ▲ Danger

By default, `iap_charge` contacts the *production* IAP endpoint, <https://iap.odoo.com> (<https://iap.odoo.com>). While developing and testing your service you may want to point it towards the *development* IAP endpoint <https://iap-sandbox.odoo.com> (<https://iap-sandbox.odoo.com>).

To do so, set the `iap.endpoint` config parameter in your service Odoo: in debug/developer mode, **Setting** ▸ **Technical** ▸ **Parameters** ▸ **System Parameters**, just define an entry for the key `iap.endpoint` if none already exists).

The `iap_charge` helper has two additional optional parameters we can use to make things clearer to the end-user.

### description

is a message which will be associated with the transaction and will be displayed in the user's dashboard, it is useful to remind the user why the charge exists.

### credit\_template

is the name of a [QWeb](#) ([../reference/qweb.html#reference-qweb](#)) template which will be rendered and shown to the user if their account has less credit available than the service provider is requesting, its purpose is to tell your users why they should be interested in your IAP offers.

*coalroller\_service/controllers/main.py*

```
def roll(self, account_token):
    # the service key is a secret, it should not be committed in
    # the source
    service_key = http.request.env['ir.config_parameter'].sudo().get_param('coalroller_service_key')

    # we charge 1 credit for 10 seconds of CPU
    cost = 1
    # TODO: allow the user to specify how many (tens of seconds) of CPU they want to
    with charge(http.request.env, service_key, account_token, cost,
                description="We're just obeying orders",
                credit_template='coalroller_service.no_credit'):

        # 10 seconds of CPU per credit
        end = time.time() + (10 * cost)
```

*coalroller\_service/views/no-credit.xml*

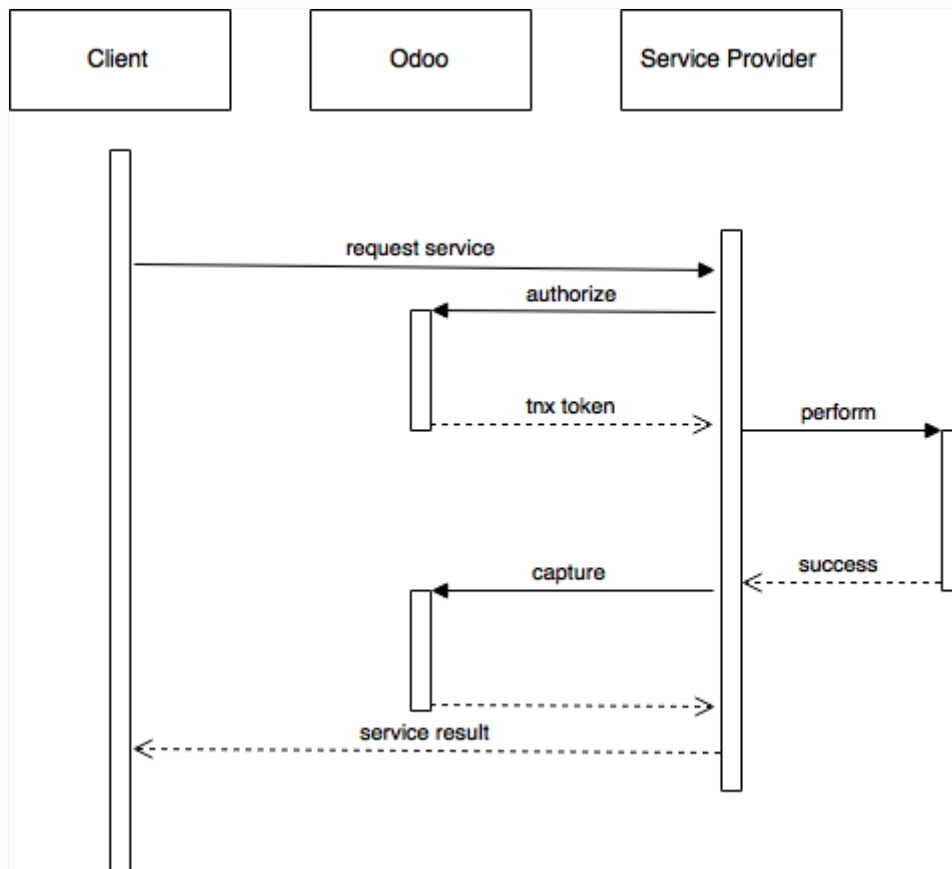
```
<odoo>
<template id="no_credit" name="No credit warning">
<div>
<div class="container-fluid">
<div class="row">
<div class="col-md-7 offset-lg-1 mt32 mb32">
<h2>Consume electricity doing nothing useful!</h2>
<ul>
<li>Heat our state of the art data center for no reason</li>
<li>Use multiple watts for only 0.1€</li>
<li>Roll coal without going outside</li>
</ul>
</div>
</div>
</div>
</div>
</template>
</odoo>
```

*coalroller\_service/\_\_manifest\_\_.py*

```
{
    'name': "Coal Roller Service",
    'category': 'Tools',
    'depends': ['iap'],
    'data': [
        'views/no-credit.xml',
    ],
}
```

## JSON-RPC2 (<https://www.jsonrpc.org/specification>) Transaction API

---



The IAP transaction API does not require using Odoo when implementing your server gateway, calls are standard JSON-RPC2 (<https://www.jsonrpc.org/specification>).

Calls use different *endpoints* but the same *method* on all endpoints ( `call` ).

Exceptions are returned as JSON-RPC2 (<https://www.jsonrpc.org/specification>) errors, the formal exception name is available on `data.name` for programmatic manipulation.

### ➔ See also

**[iap.odoo.com documentation \(https://iap.odoo.com/iap/1/documentation\)](https://iap.odoo.com/iap/1/documentation)** for additional information.

## Authorize

### **`/iap/1/authorize`**

Verifies that the user's account has at least as **credit** available *and creates a hold (pending transaction) on that amount*.

Any amount currently on hold by a pending transaction is considered unavailable to further authorize calls.

Returns a **TransactionToken** identifying the pending transaction which can be used to capture (confirm) or cancel said transaction ([iap.odoo.com documentation \(https://iap.odoo.com/iap/1/documentation\)](https://iap.odoo.com/iap/1/documentation)).

#### Parameters:

**key** ( [ServiceKey](#) ) –

**account\_token** ( [UserToken](#) ) –

**credit** ( [float](https://docs.python.org/3/library/functions.html#float) ) –

**description** ( [str](https://docs.python.org/3/library/stdtypes.html#str) ) – optional, helps users identify the reason for charges on their account

**dbuuid** ( [str](https://docs.python.org/3/library/stdtypes.html#str) ) – optional, allows the user to benefit from trial credits if his database is eligible (see [Service registration](#))

Returns:

[TransactionToken](#) if the authorization succeeded

Raises:

[AccessError](#) ([../reference/orm.html#odoo.exceptions.AccessError](#)) if the service token is invalid

Raises:

[InsufficientCreditError](#) if the account does not have enough credits

Raises:

[TypeError](#) if the **credit** value is not an integer or a float

```
r = requests.post(OD00 + '/iap/1/authorize', json={
    'jsonrpc': '2.0',
    'id': None,
    'method': 'call',
    'params': {
        'account_token': user_account,
        'key': SERVICE_KEY,
        'credit': 25,
        'description': "Why this is being charged",
    }
}).json()
if 'error' in r:
    # handle authorize error
tx = r['result']

# provide your service here
```

## Capture

### [/iap/1/capture](#)

Confirms the specified transaction, transferring the reserved credits from the user's account to the service provider's.

Capture calls are idempotent: performing capture calls on an already captured transaction has no further effect.

Parameters:

**token** ( [TransactionToken](#) ) –

**key** ( [ServiceKey](#) ) –

**credit\_to\_capture** ( [float](https://docs.python.org/3/library/functions.html#float) (<https://docs.python.org/3/library/functions.html#float>)) – optional parameter to capture a smaller amount of credits than authorized

Raises:

**[AccessError](#)** ([../reference/orm.html#odoo.exceptions.AccessError](#)).

```
r2 = requests.post(OD00 + '/iap/1/capture', json={
    'jsonrpc': '2.0',
    'id': None,
    'method': 'call',
    'params': {
        'token': tx,
        'key': SERVICE_KEY,
        'credit_to_capture': credit or False,
    }
}).json()
if 'error' in r:
    # handle capture error
# otherwise transaction is captured
```

## Cancel

**/iap/1/cancel**

Cancels the specified transaction, releasing the hold on the user's credits.

Cancel calls are idempotent: performing capture calls on an already cancelled transaction has no further effect.

Parameters:

**token** ( [TransactionToken](#) ) –

**key** ( [ServiceKey](#) ) –

Raises:

**[AccessError](#)** ([../reference/orm.html#odoo.exceptions.AccessError](#)).



```

r2 = requests.post(OD00 + '/iap/1/cancel', json={
    'jsonrpc': '2.0',
    'id': None,
    'method': 'call',
    'params': {
        'token': tx,
        'key': SERVICE_KEY,
    }
}).json()
if 'error' in r:
    # handle cancel error
# otherwise transaction is cancelled

```

## Types

Exceptions aside, these are *abstract types* used for clarity, you should not care how they are implemented.

### **class ServiceName**

String identifying your service on <https://iap.odoo.com> (<https://iap.odoo.com>) (production) as well as the account related to your service in the client's database.

### **class ServiceKey**

Identifier generated for the provider's service. Each key (and service) matches a token of a fixed value, as generated by the service provide.

Multiple types of tokens correspond to multiple services. As an example, SMS and MMS could either be the same service (with an MMS being 'worth' multiple SMS) or could be separate services at separate price points.

#### **▲ Danger**

Your service key *is a secret*, leaking your service key allows other application developers to draw credits bought for your service(s).

### **class UserToken**

Identifier for a user account.

### **class TransactionToken**

Transaction identifier, returned by the authorization process and consumed by either capturing or cancelling the transaction.

### **exception odoo.addons.iap.tools.iap\_tools.InsufficientCreditError**

Raised during transaction authorization if the credits requested are not currently available on the account (either not enough credits or too many pending transactions/existing holds).

**exception `odoo.exceptions.AccessError`**

Raised by:

any operation to which a service token is required, if the service token is invalid; or  
any failure in an inter-server call. (typically, in `iap_jsonrpc()` ).

**exception `odoo.exceptions.UserError`**

Raised by any unexpected behaviour at the discretion of the App developer (*you*).

## Test the API

In order to test the developed app, we propose a sandbox platform that allows you to:

- 1 Test the whole flow from the client's point of view - Actual services and transactions that can be consulted. (again this requires to change the endpoint, see the danger note in [Service](#)).
- 2 Test the API.

The latter consists in specific tokens that will work on **IAP-Sandbox only**.

Token **000000** : Represents a non-existing account. Returns an **InsufficientCreditError** on authorize attempt.

Token **000111** : Represents an account without sufficient credits to perform any service. Returns an **InsufficientCreditError** on authorize attempt.

Token **111111** : Represents an account with enough credits to perform any service. An authorize attempt will return a dummy transaction token that is processed by the capture and cancel routes.

Those tokens are only active on the IAP-Sandbox server.

The service key is completely ignored with this flow, If you want to run a robust test of your service, you should ignore these tokens.

## Odoo Helpers

For convenience, if you are implementing your service using Odoo the **`iap`** module provides a few helpers to make IAP flow even simpler.

### Charging

```
class odoo.addons.iap.tools.iap_tools.iap_charge(env, key, account_token, credit[, dbuuid, description, credit_template])
```

A *context manager* for authorizing and automatically capturing or cancelling transactions for use in the backend/proxy.

Works much like e.g. a cursor context manager:

- immediately authorizes a transaction with the specified parameters;
- executes the `with` body;
- if the body executes in full without error, captures the transaction;
- otherwise cancels it.

#### Parameters:

**env** ( `odoo.api.Environment` ) – used to retrieve the **iap.endpoint** configuration key

**key** ( `ServiceKey` ) –

**token** ( `UserToken` ) –

**credit** ( `float` [\\_\(https://docs.python.org/3/library/functions.html#float\)](https://docs.python.org/3/library/functions.html#float) ) –

**description** ( `str` [\\_\(https://docs.python.org/3/library/stdtypes.html#str\)](https://docs.python.org/3/library/stdtypes.html#str) ) –

**template credit\_template** ( `Qweb` ) –

```
@route('/deathstar/superlaser', type='json')
def superlaser(self, user_account,
               coordinates, target,
               factor=1.0):
    """
    :param factor: superlaser power factor,
                  0.0 is none, 1.0 is full power
    """
    credits = int(MAXIMUM_POWER * factor)
    description = "We will demonstrate the power of this station on your home planet o
    with iap_charge(request.env, SERVICE_KEY, user_account, credits, description) as t
    # TODO: allow other targets
    transaction.credit = max(credits, 2)
    # Sales ongoing one the energy price,
    # a maximum of 2 credits will be charged/captured.
    self.env['systems.planets'].search([
        ('grid', '=', 'M-10'),
        ('name', '=', 'Alderaan'),
    ]).unlink()
```

## Authorize

```
class odoo.addons.iap.tools.iap_tools.iap_authorize(env, key, account_token,
credit[, dbuuid, description, credit_template])
```

Will authorize everything.

**Parameters:**

**env** ( `odoo.api.Environment` ) – used to retrieve the **iap.endpoint** configuration key

**key** ( `ServiceKey` ) –

**token** ( `UserToken` ) –

**credit** ( `float` [\\_\(https://docs.python.org/3/library/functions.html#float\)](https://docs.python.org/3/library/functions.html#float) ) –

**description** ( `str` [\\_\(https://docs.python.org/3/library/stdtypes.html#str\)](https://docs.python.org/3/library/stdtypes.html#str) ) –

**template credit\_template** ( `Qweb` ) –

```
@route('/deathstar/superlaser', type='json')
def superlaser(self, user_account,
                coordinates, target,
                factor=1.0):
    """
    :param factor: superlaser power factor,
                  0.0 is none, 1.0 is full power
    """
    credits = int(MAXIMUM_POWER * factor)
    description = "We will demonstrate the power of this station on your home planet o
    #actual IAP stuff
    transaction_token = authorize(request.env, SERVICE_KEY, user_account, credits, des
    try:
        # Beware the power of this laser
        self.put_galactical_princess_in_sorrow()
    except Exception as e:
        # Nevermind ...
        r = cancel(env, transaction_token, key)
        raise e
    else:
        # We shall rule over the galaxy!
        capture(env, transaction_token, key, min(credits, 2))
```

## Cancel

```
class odoo.addons.iap.tools.iap_tools.iap_cancel(env, transaction_token, key)
```

Will cancel an authorized transaction.

**Parameters:**

**env** ( `odoo.api.Environment` ) – used to retrieve the **iap.endpoint** configuration key

**transaction\_token** ( `str` [\\_\(https://docs.python.org/3/library/stdtypes.html#str\)](https://docs.python.org/3/library/stdtypes.html#str) ) –

**key** ( `ServiceKey` ) –

```

@route('/deathstar/superlaser', type='json')
def superlaser(self, user_account,
                coordinates, target,
                factor=1.0):
    """
    :param factor: superlaser power factor,
                  0.0 is none, 1.0 is full power
    """
    credits = int(MAXIMUM_POWER * factor)
    description = "We will demonstrate the power of this station on your home planet o
    #actual IAP stuff
    transaction_token = authorize(request.env, SERVICE_KEY, user_account, credits, des
    try:
        # Beware the power of this laser
        self.put_galactical_princess_in_sorrow()
    except Exception as e:
        # Nevermind ...
        r = cancel(env, transaction_token, key)
        raise e
    else:
        # We shall rule over the galaxy!
        capture(env, transaction_token, key, min(credits, 2))

```

## Capture

```

class odoo.addons.iap.tools.iap_tools.iap_capture(env, transaction_token, key,
credit)

```

Will capture the amount `credit` on the given transaction.

### Parameters:

**env** ( `odoo.api.Environment` ) – used to retrieve the `iap.endpoint` configuration key

**transaction\_token** ( `str` [\\_](https://docs.python.org/3/library/stdtypes.html#str)(`https://docs.python.org/3/library/stdtypes.html#str`)) –

**key** ( `ServiceKey` ) –

**credit** –

```
@route('/deathstar/superlaser', type='json')
def superlaser(self, user_account,
                coordinates, target,
                factor=1.0):
    """
    :param factor: superlaser power factor,
                  0.0 is none, 1.0 is full power
    """
    credits = int(MAXIMUM_POWER * factor)
    description = "We will demonstrate the power of this station on your home planet o
    #actual IAP stuff
    transaction_token = authorize(request.env, SERVICE_KEY, user_account, credits, des
    try:
        # Beware the power of this laser
        self.put_galactical_princess_in_sorrow()
    except Exception as e:
        # Nevermind ...
        r = cancel(env, transaction_token, key)
        raise e
    else:
        # We shall rule over the galaxy!
        capture(env, transaction_token, key, min(credits, 2))
```