Using the Core API in Python

Python

The Core API is based on HTTP and OAuth and provides low-level calls to access and manipulate a user's Dropbox account.

If you want to follow along, first register a new app on the App Console. You'll need the app key to access the Core API. Then install the Python SDK and you'll be ready to go.

Authenticating your app

The Core API uses OAuth v2, but the Python SDK will take care of most of it so you don't have to start from scratch.

You'll need to provide your app key and secret to the new `DropboxOAuth2FlowNoRedirect` object.

```
# Include the Dropbox SDK
import dropbox

# Get your app key and secret from the Dropbox developer website
app_key = 'INSERT_APP_KEY'
app_secret = 'INSERT_APP_SECRET'

flow = dropbox.client.DropboxOAuth2FlowNoRedirect(app_key, app_secret)
```

Now we're all set to start the OAuth flow. The OAuth flow has two parts:

1. Ask the user to authorize linking your app to their Dropbox account.
2. Once authorized, exchange the received authorization code for an access token, which will be used for calling the Core API.

We'll start by using the `DropboxOAuth2FlowNoRedirect` object to generate an authorization URL with the `start` method.

```
authorize_url = flow.start()
```

With the authorization URL in hand, we can now ask the user to authorize your app. To avoid the hassle of setting up a web server in this tutorial, we're just printing the URL and asking the user to press the Enter key to confirm that they've authorized your app. However, in real-world apps, you'll want to automatically send the user to the authorization URL and pass in a callback URL so that the user is seamlessly redirected back to your app after pressing a button.

```
# Have the user sign in and authorize this token
authorize_url = flow.start()
print '1. Go to: ' + authorize_url
```

```
print '2. Click "Allow" (you might have to log in first)'
print '3. Copy the authorization code.'
code = raw_input("Enter the authorization code here: ").strip()
```

Once the user has delivered the authorization code to our app, we can exchange that code for an access token via `finish`:

```
# This will fail if the user enters an invalid authorization code
access_token, user_id = flow.finish(code)
```

The access token is all you'll need to make API requests on behalf of this user, so you should store it away for safe-keeping (even though we don't for this tutorial). By storing the access token, you won't need to go through these steps again unless the user reinstalls your app or revokes access via the Dropbox website.

Now that the hard part is done, all we need to do to authorize our API calls is to to pass the access token to `DropboxClient`. To test that we have got access to the Core API, let's try calling `account_info`, which will return a dictionary with information about the user's linked account:

```
client = dropbox.client.DropboxClient(access_token)
print 'linked account: ', client.account_info()
```

If you've made it this far, you now have a simple app that uses the Core API to link to a Dropbox account and make an API call to retrieve account info. Next, we'll upload a file to Dropbox, get its metadata, and then download it back to our app.

Uploading files

Let's say we're building a text editing app and we want to use it to save your latest magnum opus to Dropbox. Let's browse the methods in the Python docs to see which one will do that for us. This page lists all the methods supported in the SDK. If you scroll down, you'll find `put_file`.

`put_file` takes a path pointing to where we want the file on our Dropbox, and then a file-like object or string to be uploaded there. For this example, let's upload a local copy of **working-draft.txt**:

```
f = open('working-draft.txt', 'rb')
response = client.put_file('/magnum-opus.txt', f)
print "uploaded:", response
```

If all goes well, the data in your local **working-draft.txt** will now be in the root of your app folder (or Dropbox folder, depending on your app's access type). The variable `response` will be a dictionary containing the metadata of the newly uploaded file. It will look something like this:

```
{
    'bytes': 77,
    'icon': 'page_white_text',
    'is_dir': False,
    'mime_type': 'text/plain',
    'modified': 'Wed, 20 Jul 2011 22:04:50 +0000',
    'path': '/magnum-opus.txt',
    'rev': '362e2029684fe',
    'revision': 221922,
    'root': 'dropbox',
    'size': '77 bytes',
    'thumb_exists': False
}
```

Listing folders

If you peruse the various entries in the metadata above, you'll get a good sense for all info we can gather from the file. You can get this info for an entire folder by using the `metadata` call.

```
folder_metadata = client.metadata('/')
print "metadata:", folder_metadata
```

The returned dictionary will list out the files and folders in the path given. It looks something like this:

```
{'bytes': 0,
 'contents': [{'bytes': 0,
               'icon': 'folder',
               'is_dir': True,
               'modified': 'Thu, 25 Aug 2011 00:03:15 +0000',
               'path': '/Sample Folder',
               'rev': '803beb471',
               'revision': 8,
               'root': 'dropbox',
               'size': '0 bytes',
               'thumb_exists': False},
              {'bytes': 77,
               'icon': 'page_white_text',
               'is_dir': False,
               'mime_type': 'text/plain',
               'modified': 'Wed, 20 Jul 2011 22:04:50 +0000',
               'path': '/magnum-opus.txt',
               'rev': '362e2029684fe',
               'revision': 221922,
               'root': 'dropbox',
               'size': '77 bytes',
               'thumb_exists': False}],
 'hash': 'efdac89c4da886a9cece1927e6c22977',
 'icon': 'folder',
 'is_dir': True,
 'path': '/',
 'root': 'app_folder',
```

```
 'size': '0 bytes',
 'thumb_exists': False}
```

In the example above, the app folder root contains a directory named **Sample Folder** and the file we just uploaded named **magnum-opus.txt**. You'll also see other various but vital bits of information such as the exact location of the file (`path`), file sizes (`bytes`), last modified date (`modified`), and more. If you want to tell if something has changed in the directory, you'll want to store and compare the `hash` parameter. Similarly, a file's `rev` parameter tells you if the file has changed. It's useful to keep track of the status of your files, as we'll see in the following example.

Downloading files

Some time has passed and you're ready to start editing that magnum opus of yours again. We'll need the `get_file_and_metadata` method to download the file.

```
f, metadata = client.get_file_and_metadata('/magnum-opus.txt')
out = open('magnum-opus.txt', 'wb')
out.write(f.read())
out.close()
print metadata
```

`get_file_and_metadata`, like other calls that return file data, returns an `httplib.HTTPResponse` that you should `.read()` from to get the full response.

In addition to the file, the method also returns the file's metadata at its current revision. Every time a change is made to the file, the `rev` field of the file's metadata changes as well. By saving the revision when you download the file, you'll be able to tell if that file has been updated by another computer or device and choose to download the newer revision of that file.

The complete code

For those keeping score at home, here's the full source to this guide. Make sure to create a **working-draft.txt** file to get it to work fully. Also remember to insert your app key, app secret, and access type.

```
# Include the Dropbox SDK
import dropbox

# Get your app key and secret from the Dropbox developer website
app_key = 'INSERT_APP_KEY'
app_secret = 'INSERT_APP_SECRET'

flow = dropbox.client.DropboxOAuth2FlowNoRedirect(app_key, app_secret)

# Have the user sign in and authorize this token
authorize_url = flow.start()
print '1. Go to: ' + authorize_url
print '2. Click "Allow" (you might have to log in first)'
print '3. Copy the authorization code.'
code = raw_input("Enter the authorization code here: ").strip()
```

```
# This will fail if the user enters an invalid authorization code
access_token, user_id = flow.finish(code)

client = dropbox.client.DropboxClient(access_token)
print 'linked account: ', client.account_info()

f = open('working-draft.txt', 'rb')
response = client.put_file('/magnum-opus.txt', f)
print 'uploaded: ', response

folder_metadata = client.metadata('/')
print 'metadata: ', folder_metadata

f, metadata = client.get_file_and_metadata('/magnum-opus.txt')
out = open('magnum-opus.txt', 'wb')
out.write(f.read())
out.close()
print metadata
```

Python SDK Documentation

You can also dive into the Python SDK documentation, or if you're feeling ambitious, browse the Python SDK source code. The source code is found in the folder labeled **dropbox** within the SDK package. The file you'll be interfacing the most with is **client.py**. You won't need to deal with **rest.py** unless you intend to implement your own calls, but it's responsible for actually making API requests and parsing the server's response.

Next steps

And with that you should be equipped with most everything you need to get started with the Core API. If you're still not sure about something, the developer forum is a great place to find information and get help from fellow developers. Good luck!