datetime

A Tale of Pythonic Woe

Dave Voutila

voutilad@gmail.com
https://github.com/voutilad
@voutilad

23 August 2017

Who am I?

- Independent Software Consultant
- Start-up Advisor
- Specializing in
  - Python & Java development expertise
  - Software go-to-market & Sales
  - Making PowerPoint slides
  - Wasting time with LaTeX

Sisu
integrated services llc

Task Analytics

What's with the little people?

Figure: Time Mages — Final Fantasy Tactics

Once upon a Github Issue...

Flask-Ask

Rapid Alexa Skills Kit Development
for Amazon Echo Devices

- Contributor to **Flask-Ask**, a Python Amazon Alexa framework.
- Jumped on *Issue 152: Flask Ask doesn't parse time stamp from Alexa properly...*
- Things went downhill from there...

A Classic Github Issue

Flask Ask doesn't parse time stamp from Alexa properly. #152

Closed **robputt796** opened this issue on Jul 8 · 44 comments

robputt796 commented on Jul 8

```
Traceback (most recent call last):
File "/opt/HomeApp/lib/python3.4/site-packages/flask/app.py", line 1997, in call
return self.wsgi_app(environ, start_response)
File "/opt/HomeApp/lib/python3.4/site-packages/flask/app.py", line 1985, in wsgi_app
response = self.handle_exception(e)
File "/opt/HomeApp/lib/python3.4/site-packages/flask/app.py", line 1540, in handle_exception
reraise(exc_type, exc_value, tb)
File "/opt/HomeApp/lib/python3.4/site-packages/flask/_compat.py", line 33, in reraise
raise value
File "/opt/HomeApp/lib/python3.4/site-packages/flask/app.py", line 1982, in wsgi_app
response = self.full_dispatch_request()
```

- Seriously...just a Traceback
- `AttributeError: 'int' object has no attribute 'split'`

The Offending Code

```
timestamp = aniso8601.parse_datetime(
        alexa_request_payload['request']['timestamp']
)
```

- `aniiso8601` — ISO-8601 parsing library
- Trying to de-reference items in the Alexa JSON
- `aniiso8601` is not happy with the value it's given
- So if it's not a String, what is it?

RTFM

- Let's see what Amazon's documentation says about "timestamps":

> The `timestamp` is provided as an ISO 8601 formatted string (for example, 2015-05-13T12:34:56Z). Your code needs to parse the string into a date object, then verify that it is within the tolerance your web service allows (no more than 150 seconds). Reject requests in which the `timestamp` falls outside the tolerance with an error code (such as `400 Bad Request`).

Figure: `https://developer.amazon.com/public/solutions/alexa/alexa-skills-kit/docs/developing-an-alexa-skill-as-a-web-service#timestamp`

- Ok, so we *should* get something like: `"2009-02-13T23:31:30Z"`

Alexa

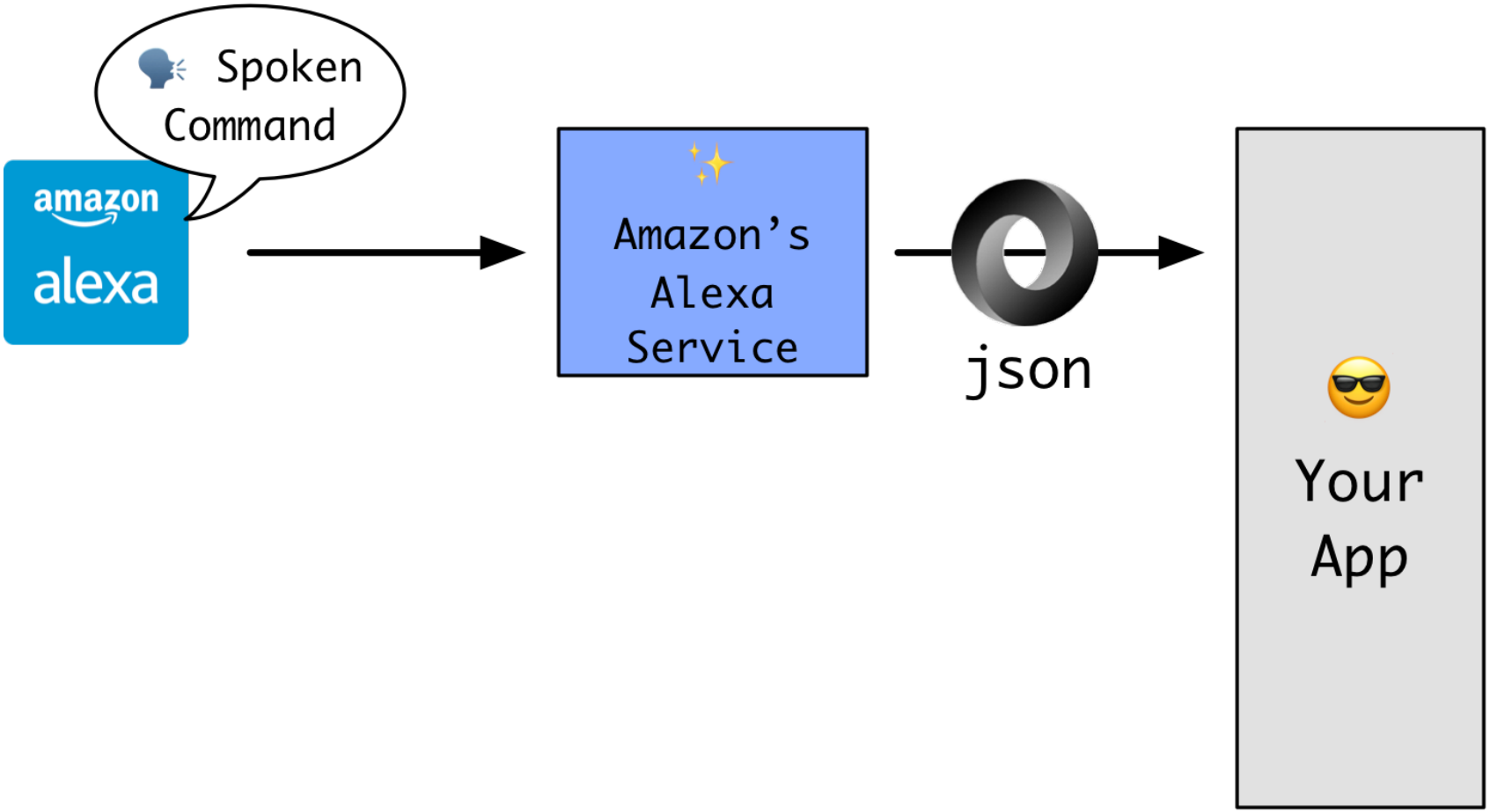

Figure: super simple Alexa architecture

Amazon Lies

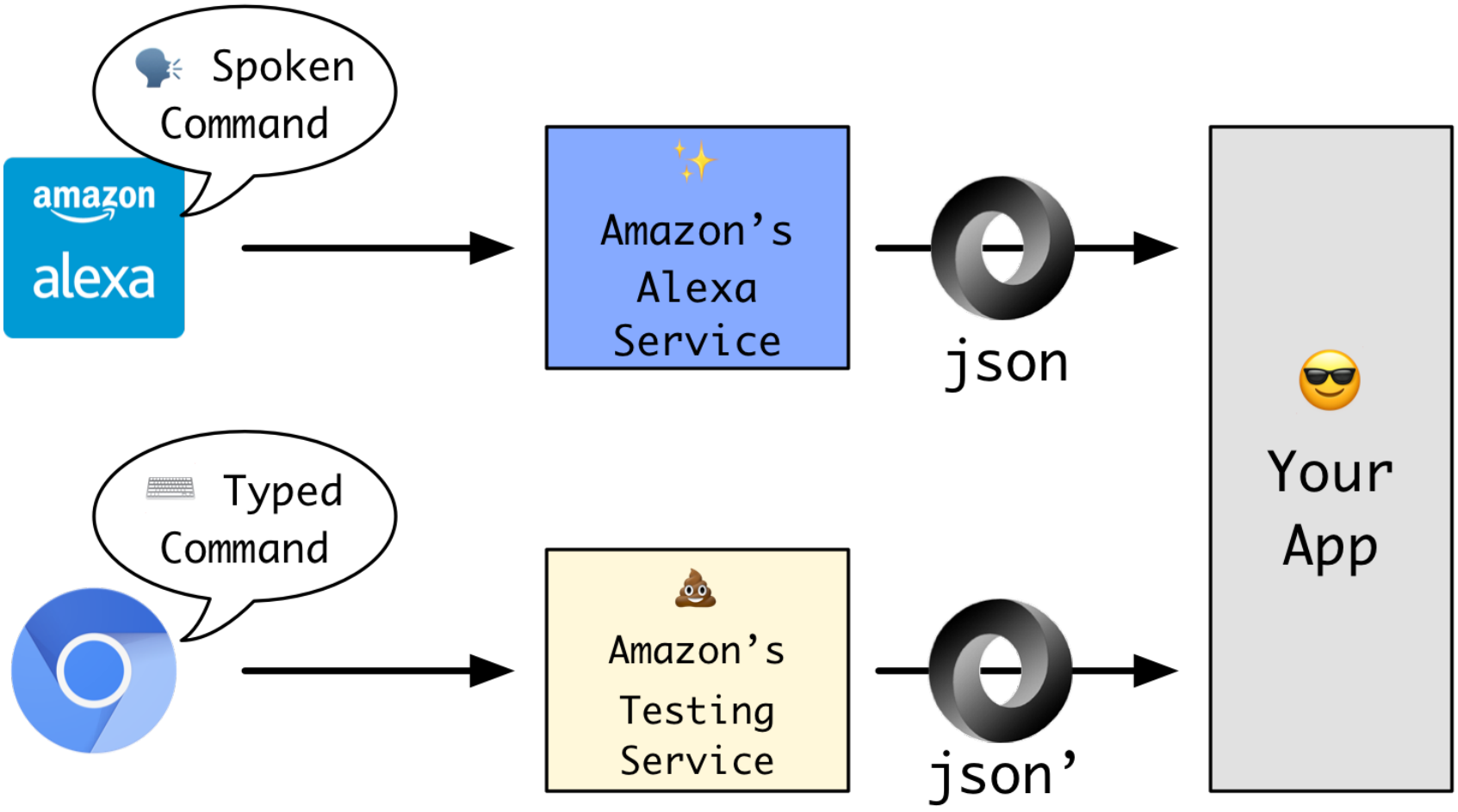

Figure: json $\neq$ json'

Good Agent Cooper vs. Evil Agent Cooper

- One of these things is not like the other...

```
{
  "request": {
    "type": "LaunchRequest",
    "requestId": "<guid>",
    "locale": "en-US",
    "timestamp":
↪    "2009-02-13T23:31:30Z"
  }
}
```

```
{
  "request": {
    "type": "LaunchRequest",
    "requestId": "<guid>",
    "locale": "en-US",
    "timestamp":
↪    1234567890000
  }
}
```

- So, that lovely integer is the epoch time in milliseconds.
- ...for UTC? ¯_(ツ)_/¯

First Attempt

- Wanted a simple solution to handle both `strings` and `ints`
- If fails to parse, handle the `AttributeError` and assume an int

```
552     @staticmethod
553     def _parse_timestamp(timestamp):
554         """
555         Parse a given timestamp value, raising ValueError if None or Flasey
556         """
557         if timestamp:
558             try:
559                 return aniso8601.parse_datetime(timestamp)
560             except AttributeError:
561                 # raised by aniso8601 if raw_timestamp is not valid string in ISO8601 format
562                 return datetime.utcfromtimestamp(timestamp)
563 
564         raise ValueError('Invalid timestamp value! Cannot parse from either ISO8601 string or UTC timestamp.')
```

Figure: core.py — 3240a43c4dce6b1cf45754c2d8ac82a7f9c150a6

Lesson 1: Python's a bit Odd (as is C#)

- Well...it's not just Python, but precision is key.

Language	Example	Precision
Go	`time.Time`	nanoseconds
Java	`java.lang.System.getNanos()`	nanoseconds
C#	`DateTime.Ticks`	$\frac{1}{10}$ microseconds
Javascript	`Date.now()`	milliseconds
Python	`time.time()`	microseconds

- But wait, there's more!

Nobody will live to see 41091 AD anyways...

```
Python 3.6.2 (default, Jul 17 2017, 16:44:45)
[GCC 4.2.1 Compatible Apple LLVM 8.1.0 (clang-802.0.42)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from datetime import datetime
>>> datetime.utcfromtimestamp(1234567890000)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: year 41091 is out of range
```

- Ok, so if Amazon sends milliseconds, lets scale it.

Second Attempt

- So turns out Python uses **microseconds**
- Try this again, but now while scaling the value...

```
558  558          try:
559  559              return aniso8601.parse_datetime(timestamp)
560  560          except AttributeError:
561       -          # raised by aniso8601 if raw_timestamp is not valid string in ISO8601 format
562       -          return datetime.utcfromtimestamp(timestamp)
     561  +          # raised by aniso8601 if raw_timestamp is not valid string
     562  +          # in ISO8601 format
     563  +          try:
     564  +              return datetime.utcfromtimestamp(timestamp)
     565  +          except ValueError:
     566  +              # relax the timestamp a bit in case it was sent in millis
     567  +              return datetime.utcfromtimestamp(timestamp/1000)
563  568
564  569      raise ValueError('Invalid timestamp value! Cannot parse from either ISO8601 string or UTC timestamp.')
```

Figure: core.py — 17ba43a60fc4e0b91f00d596aa3cfc78c81771a9

Never test on just your machine!

- Windows: what the heck ¯_(ツ)_/¯

```
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:53:40) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from datetime import datetime
>>> datetime.utcfromtimestamp(1234567890000)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: timestamp out of range for platform localtime()/gmtime() function
>>>
```

```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:18:55) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from datetime import datetime
>>> datetime.utcfromtimestamp(1234567890000)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
OSError: [Errno 22] Invalid argument
>>>
```

Lesson 2: Python Time Handling is System Dependent

- Python's `time` module is all native `C`
- `datetime` uses `time`
  - `datetime` is pure Python
  - A leaky abstraction
- Calls to `datetime.utcfromtimestamp()` trigger the code on the right
- **Hint** — this will be a factor

```
790  int
791  _PyTime_localtime(time_t t, struct tm *tm)
792  {
793  #ifdef MS_WINDOWS
794      int error;
795
796      error = localtime_s(tm, &t);
797      if (error != 0) {
798          errno = error;
799          PyErr_SetFromErrno(PyExc_OSError);
800          return -1;
801      }
802      return 0;
803  #else /* !MS_WINDOWS */
804      if (localtime_r(&t, tm) == NULL) {
805  #ifdef EINVAL
806          if (errno == 0)
807              errno = EINVAL;
808  #endif
809          PyErr_SetFromErrno(PyExc_OSError);
810          return -1;
811      }
812      return 0;
813  #endif /* MS_WINDOWS */
814  }
```

Figure: pytime.c

Third Attempt

- I give up!

```
562  562                 # in ISO8601 format
563  563                 try:
564  564                     return datetime.utcfromtimestamp(timestamp)
565       -             except ValueError:
     565   +             except:
566  566                     # relax the timestamp a bit in case it was sent in millis
567  567                     return datetime.utcfromtimestamp(timestamp/1000)
568  568
```

Figure: core.py — 8de0db687cb6ff28d7b9bed251480060b01d5736

Happy ending?

- Case closed! I guess it works now.
- But...Python still thinks none of us will like to see 10000AD :-(

```
40      #define MINYEAR 1
41      #define MAXYEAR 9999
```

Figure: CPython's _datetimemodule.c