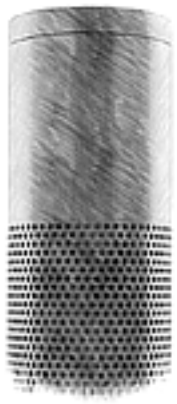
Flask-Ask

Aug 13, 2018

Contents



Flask-Ask

Rapid Alexa Skills Kit Development for Amazon Echo Devices

Lighten your cognitive load. Level up with the Alexa Skills Kit Video Tutorial.

Building high-quality Alexa skills for Amazon Echo Devices takes time. Flask-Ask makes it easier and much more fun. Use Flask-Ask with ngrok to eliminate the deploy-to-test step and get work done faster.

Flask-Ask:

- Has decorators to map Alexa requests and intent slots to view functions
- Helps construct ask and tell responses, reprompts and cards
- Makes session management easy
- Allows for the separation of code and speech through Jinja templates
- Verifies Alexa request signatures

Follow along with this quickstart on Amazon.

CHAPTER 1

Table Of Contents

1.1 Getting Started

1.1.1 Installation

To install Flask-Ask:

```
pip install flask-ask
```

1.1.2 A Minimal Voice User Interface

A Flask-Ask application looks like this:

```
from flask import Flask, render_template
from flask_ask import Ask, statement

app = Flask(__name__)
ask = Ask(app, '/')

@ask.intent('HelloIntent')
def hello(firstname):
    text = render_template('hello', firstname=firstname)
    return statement(text).simple_card('Hello', text)

if __name__ == '__main__':
    app.run(debug=True)
```

In the code above:

1. The `Ask` object is created by passing in the Flask application and a route to forward Alexa requests to.
2. The `intent` decorator maps `HelloIntent` to a view function `hello`.
3. The intent's `firstname` slot is implicitly mapped to `hello`'s `firstname` parameter.

4. Jinja templates are supported. Internally, templates are loaded from a YAML file (discussed further below).
5. Lastly, a builder constructs a spoken response and displays a contextual card in the Alexa smartphone/tablet app.

Since Alexa responses are usually short phrases, it's convenient to put them in the same file. Flask-Ask has a Jinja template loader that loads multiple templates from a single YAML file. For example, here's a template that supports the minimal voice interface above.Templates are stored in a file called *templates.yaml* located in the application root:

```
hello: Hello, {{ firstname }}
```

For more information about how the Alexa Skills Kit works, see Understanding Custom Skills in the Alexa Skills Kit documentation.

Additionally, more code and template examples are in the samples directory.

1.2 Handling Requests

With the Alexa Skills Kit, spoken phrases are mapped to actions executed on a server. Alexa converts speech into JSON and delivers the JSON to your application. For example, the phrase:

> "Alexa, Tell HelloApp to say hi to John"

produces JSON like the following:

```
"request": {
  "intent": {
    "name": "HelloIntent",
    "slots": {
      "firstname": {
        "name": "firstname",
        "value": "John"
      }
    }
  }
  ...
}
```

Parameters called 'slots' are defined and parsed out of speech at runtime. For example, the spoken word 'John' above is parsed into the slot named `firstname` with the `AMAZON.US_FIRST_NAME` data type.

For detailed information, see Handling Requests Sent by Alexa on the Amazon developer website.

This section shows how to process Alexa requests with Flask-Ask. It contains the following subsections:

1.2.1 Mapping Alexa Requests to View Functions

Here is a video demo on Handling Requests with Flask-Ask video.

Flask-Ask has decorators to map Alexa requests to view functions.

The `launch` decorator handles launch requests:

```
@ask.launch
def launched():
    return question('Welcome to Foo')
```

The `intent` decorator handles intent requests:

```
@ask.intent('HelloWorldIntent')
def hello():
    return statement('Hello, world')
```

The `session_ended` decorator is for the session ended request:

```
@ask.session_ended
def session_ended():
    return "{}", 200
```

Launch and intent requests can both start sessions. Avoid duplicate code with the `on_session_started` callback:

```
@ask.on_session_started
def new_session():
    log.info('new session started')
```

1.2.2 Mapping Intent Slots to View Function Parameters

Here is a video demo on Intent Slots with Flask-Ask video.

When Parameter and Slot Names Differ

Tell Flask-Ask when slot and view function parameter names differ with `mapping`:

```
@ask.intent('WeatherIntent', mapping={'city': 'City'})
def weather(city):
    return statement('I predict great weather for {}'.format(city))
```

Above, the parameter `city` is mapped to the slot `City`.

Assigning Default Values when Slots are Empty

Parameters are assigned a value of `None` if the Alexa service:

- Does not return a corresponding slot in the request
- Includes a corresponding slot without its `value` attribute
- Includes a corresponding slot with an empty `value` attribute (e.g. `""`)

Use the `default` parameter for default values instead of `None`. The default itself should be a literal or a callable that resolves to a value. The next example shows the literal `'World'`:

```
@ask.intent('HelloIntent', default={'name': 'World'})
def hello(name):
    return statement('Hello, {}'.format(name))
```

Converting Slots Values to Python Data Types

Here is a video demo on Slot Conversions with Flask-Ask video.

When slot values are available, they're always assigned to parameters as strings. Convert to other Python data types with `convert`. `convert` is a `dict` that maps parameter names to callables:

```
@ask.intent('AddIntent', convert={'x': int, 'y': int})
def add(x, y):
    z = x + y
    return statement('{} plus {} equals {}'.format(x, y, z))
```

Above, `x` and `y` will both be passed to `int()` and thus converted to `int` instances.

Flask-Ask provides convenient API constants for Amazon `AMAZON.DATE`, `AMAZON.TIME`, and `AMAZON.DURATION` types exist since those are harder to build callables against. Instead of trying to define functions that work with inputs like those in Amazon's documentation, just pass the strings in the second column below:

Here is a video demo on Slot Conversion Helpers with Flask-Ask video.

Amazon Data Type	String	Python Data Type
`AMAZON.DATE`	`'date'`	`datetime.date`
`AMAZON.TIME`	`'time'`	`datetime.time`
`AMAZON.DURATION`	`'timedelta'`	`datetime.timedelta`

Examples

```
convert={'the_date': 'date'}
```

converts `'2015-11-24'`, `'2015-W48-WE'`, or `'201X'` into a `datetime.date`

```
convert={'appointment_time': 'time'}
```

converts `'06:00'`, `'14:15'`, or `'23:59'` into a `datetime.time`.

```
convert={'ago': 'timedelta'}
```

converts `'PT10M'`, `'PT45S'`, or `'P2YT3H10M'` into a `datetime.timedelta`.

Handling Conversion Errors

Sometimes Alexa doesn't understand what's said, and slots come in with question marks:

```
"slots": {
  "age": {
    "name": "age",
    "value": "?"
  }
}
```

Recover gracefully with the `convert_errors` context local. Import it to use it:

```
...
from flask_ask import statement, question, convert_errors


@ask.intent('AgeIntent', convert={'age': int})
def say_age(age):
    if 'age' in convert_errors:
        # since age failed to convert, it keeps its string
        # value (e.g. "?") for later interrogation.
        return question("Can you please repeat your age?")

    # conversion guaranteed to have succeeded
    # age is an int
    return statement("Your age is {}".format(age))
```

`convert_errors` is a `dict` that maps parameter names to the `Exceptions` raised during conversion. When writing your own converters, raise `Exceptions` on failure, so they work with `convert_errors`:

```
def to_direction_const(s):
    if s.lower() not in ['left', 'right']
        raise Exception("must be left or right")
    return LEFT if s == 'left' else RIGHT

@ask.intent('TurnIntent', convert={'direction': to_direction_const})
def turn(direction):
    # do something with direction
    ...
```

That `convert_errors` is a `dict` allows for granular error recovery:

```
if 'something' in convert_errors:
    # Did something fail?
```

or:

```
if convert_errors:
    # Did anything fail?
```

1.2.3 `session`, `context`, `request` and `version` Context Locals

An Alexa request payload has four top-level elements: `session`, `context`, `request` and `version`. Like Flask, Flask-Ask provides context locals that spare you from having to add these as extra parameters to your functions. However, the `request` and `session` objects are distinct from Flask's `request` and `session`. Flask-Ask's `request`, `context` and `session` correspond to the Alexa request payload components while Flask's correspond to lower-level HTTP constructs.

To use Flask-Ask's context locals, just import them:

```
from flask import App
from flask_ask import Ask, request, context, session, version

app = Flask(__name__)
ask = Ask(app)
log = logging.getLogger()
```

(continues on next page)

(continued from previous page)

```
@ask.intent('ExampleIntent')
def example():
    log.info("Request ID: {}".format(request.requestId))
    log.info("Request Type: {}".format(request.type))
    log.info("Request Timestamp: {}".format(request.timestamp))
    log.info("Session New?: {}".format(session.new))
    log.info("User ID: {}".format(session.user.userId))
    log.info("Alexa Version: {}".format(version))
    log.info("Device ID: {}".format(context.System.device.deviceId))
    log.info("Consent Token: {}".format(context.System.user.permissions.consentToken))
    ...
```

If you want to use both Flask and Flask-Ask context locals in the same module, use `import as`:

```
from flask import App, request, session
from flask_ask import (
    Ask,
    request as ask_request,
    session as ask_session,
    version
)
```

For a complete reference on `request`, `context` and `session` fields, see the JSON Interface Reference for Custom Skills in the Alexa Skills Kit documentation.

1.3 Building Responses

Here is a video demo on Building Responses with Flask-Ask video .

The two primary constructs in Flask-Ask for creating responses are `statement` and `question`.

Statements terminate Echo sessions. The user is free to start another session, but Alexa will have no memory of it (unless persistence is programmed separately on the server with a database or the like).

A `question`, on the other hand, prompts the user for additional speech and keeps a session open. This session is similar to an HTTP session but the implementation is different. Since your application is communicating with the Alexa service instead of a browser, there are no cookies or local storage. Instead, the session is maintained in both the request and response JSON structures. In addition to the session component of questions, questions also allow a `reprompt`, which is typically a rephrasing of the question if user did not answer the first time.

This sections shows how to build responses with Flask-Ask. It contains the following subsections:

1.3.1 Telling with `statement`

`statement` closes the session:

```
@ask.intent('AllYourBaseIntent')
def all_your_base():
    return statement('All your base are belong to us')
```

1.3.2 Asking with `question`

Asking with `question` prompts the user for a response while keeping the session open:

```
@ask.intent('AppointmentIntent')
def make_appointment():
    return question("What day would you like to make an appointment for?")
```

If the user doesn't respond, encourage them by rephrasing the question with `reprompt`:

```
@ask.intent('AppointmentIntent')
def make_appointment():
    return question("What day would you like to make an appointment for?") \
      .reprompt("I didn't get that. When would you like to be seen?")
```

1.3.3 Session Management

The `session` context local has an `attributes` dictionary for persisting information across requests:

```
session.attributes['city'] = "San Francisco"
```

When the response is rendered, the session attributes are automatically copied over into the response's `sessionAttributes` structure.

The renderer looks for an `attribute_encoder` on the session. If the renderer finds one, it will pass it to `json.dumps` as either that function's `cls` or `default` keyword parameters depending on whether a `json.JSONEncoder` or a function is used, respectively.

Here's an example that uses a function:

```
def _json_date_handler(obj):
    if isinstance(obj, datetime.date):
        return obj.isoformat()

session.attributes['date'] = date
session.attributes_encoder = _json_date_handler
```

See the json.dump documentation for for details about that method's `cls` and `default` parameters.

1.3.4 Automatic Handling of Plaintext and SSML

The Alexa Skills Kit supports plain text or SSML outputs. Flask-Ask automatically detects if your speech text contains SSML by attempting to parse it into XML, and checking if the root element is `speak`:

```
try:
    xmldoc = ElementTree.fromstring(text)
    if xmldoc.tag == 'speak':
        # output type is 'SSML'
except ElementTree.ParseError:
    pass
# output type is 'PlainText'
```

1.3.5 Displaying Cards in the Alexa Smartphone/Tablet App

In addition to speaking back, Flask-Ask can display contextual cards in the Alexa smartphone/tablet app. All four of the Alexa Skills Kit card types are supported.

Simple cards display a title and message:

```
@ask.intent('AllYourBaseIntent')
def all_your_base():
    return statement('All your base are belong to us') \
      .simple_card(title='CATS says...', content='Make your time')
```

Standard cards are like simple cards but they also support small and large image URLs:

```
@ask.intent('AllYourBaseIntent')
def all_your_base():
    return statement('All your base are belong to us') \
        .standard_card(title='CATS says...',
                       text='Make your time',
                       small_image_url='https://example.com/small.png',
                       large_image_url='https://example.com/large.png')
```

Link account cards display a link to authorize the Alexa user with a user account in your system. The link displayed is the auhorization URL you configure in the amazon skill developer portal:

```
@ask.intent('AllYourBaseIntent')
def all_your_base():
    return statement('Please link your account in the Alexa app') \
    .link_account_card()
```

Consent cards ask for the permission to access the device's address. You can either ask for the country and postal code (*read::alexa:device:all:address:country_and_postal_code*) or for the full address (*read::alexa:device:all:address*). The permission you ask for has to match what you've specified in the amazon skill developer portal:

```
@ask.intent('AllYourBaseIntent')
def all_your_base():
    return statement('Please allow access to your location') \
    .consent_card("read::alexa:device:all:address")
```

1.3.6 Jinja Templates

You can also use Jinja templates. Define them in a YAML file named *templates.yaml* inside your application root:

```
@ask.intent('RBelongToUsIntent')
def all_your_base():
```

(continues on next page)

(continued from previous page)

```
    notice = render_template('all_your_base_msg', who='us')
    return statement(notice)
```

```
all_your_base_msg: All your base are belong to {{ who }}

multiple_line_example: |
  <speak>
      I am a multi-line SSML template. My content spans more than one line,
      so there's a pipe and a newline that separates my name and value.
      Enjoy the sounds of the ocean.
      <audio src='https://s3.amazonaws.com/ask-storage/tidePooler/OceanWaves.mp3'/>
  </speak>
```

You can also use a custom templates file passed into the Ask object:

```
ask = Ask(app, '/', None, 'custom-templates.yml')
```

1.4 Configuration

1.4.1 Configuration

Flask-Ask exposes the following configuration variables:

ASK_APPLICATION_ID	Turn on application ID verification by setting this variable to an application ID or a list of allowed application IDs. By default, application ID verification is disabled and a warning is logged. This variable should be set in production to ensure requests are being sent by the applications you specify. **Default:** `None`
ASK_VERIFY_REQUESTS	Enables or disables Alexa request verification, which ensures requests sent to your skill are from Amazon's Alexa service. This setting should not be disabled in production. It is useful for mocking JSON requests in automated tests. **Default:** `True`
ASK_VERIFY_TIMESTAMP_DEBUG	Turn on request timestamp verification while debugging by setting this to `True`. Timestamp verification helps mitigate against replay attacks. It relies on the system clock being synchronized with an NTP server. This setting should not be enabled in production. **Default:** `False`

1.4.2 Logging

To see the JSON request / response structures pretty printed in the logs, turn on `DEBUG`-level logging:

```
import logging

logging.getLogger('flask_ask').setLevel(logging.DEBUG)
```

1.5 User Contributions

Have an article or video to submit? Please send it to john@johnwheeler.org

Flask-Ask: A New Python Framework for Rapid Alexa Skills Kit Development

by John Wheeler

Running with Alexa Part I.

by Tim Kjær Lange

Intro and Skill Logic - Alexa Skills w/ Python and Flask-Ask Part 1

by Harrison Kinsley

Headlines Function - Alexa Skills w/ Python and Flask-Ask Part 2

by Harrison Kinsley

Testing our Skill - Alexa Skills w/ Python and Flask-Ask Part 3

by Harrison Kinsley

Flask-Ask—A tutorial on a simple and easy way to build complex Alexa Skills

by Bjorn Vuylsteker