asn1.io/ asn1-python-compiler

simple yet versatile

by OSS NOKALVA, INC.

Summary: ASN.1 End-to-End

- ASN.1 schema to Python codec: simple, versatile, and fast
- End-to-End tools: schema authoring, documentation, codec generation, data troubleshooting
- 5-step quick start: get schema, specify codec, generate library, code, send/receive data
- Power user tips: upload multiple schema files, optimize with OER, minimize code size, use class-based bindings, generate random test data
- Simple to code: learn 3 functions (encode, decode, validate), manipulate data with dictionaries or classes
- Two binding options: native (dictionaries) or class-based (objects)

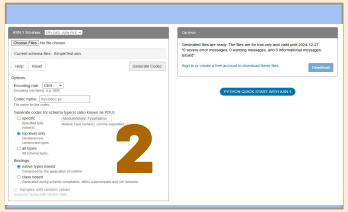
ASN.1 End-to-End

Beyond the ASN.1 Python compiler, OSS offers tools for every step of your ASN.1 journey:

- Schema authoring tools and support of the most comprehensive ASN.1 features on the market
  Analyzer, Documenter, VS Code Extension
- Generated Python codec library specifically for your schema and your protocol
  two kinds of bindings (dictionaries and classes), optimized for speed, choice of binary format
  BER, DER, OER, COER, PER, UPER, CPER, CUPER
- ASN.1 data troubleshooting tools
  Playgrounds, Inspectors

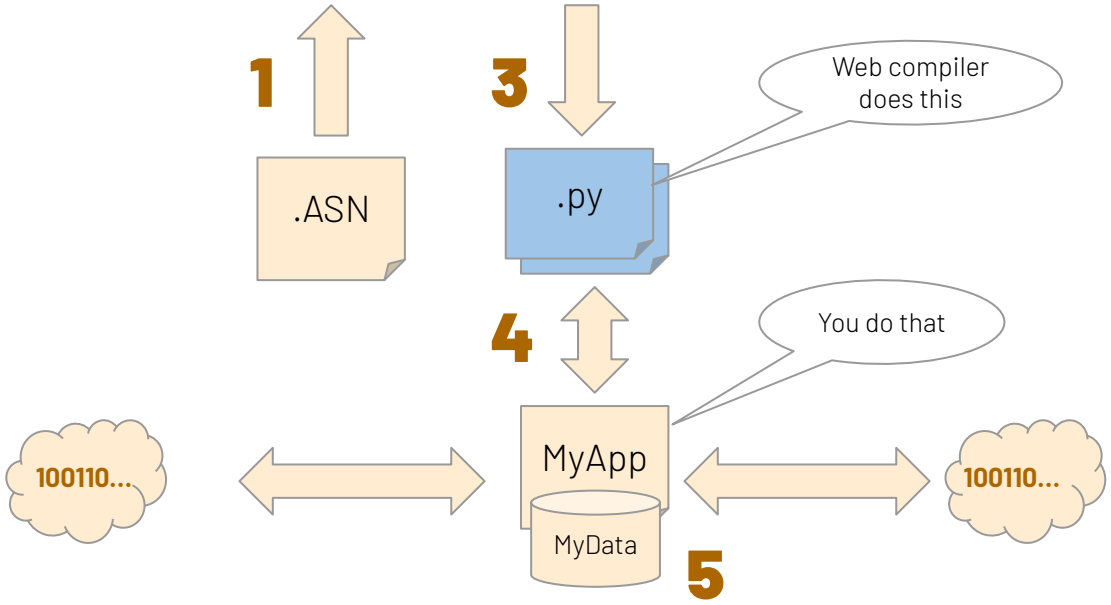
Ask us for more details info@oss.com

Quick start



You need to communicate in ASN.1:

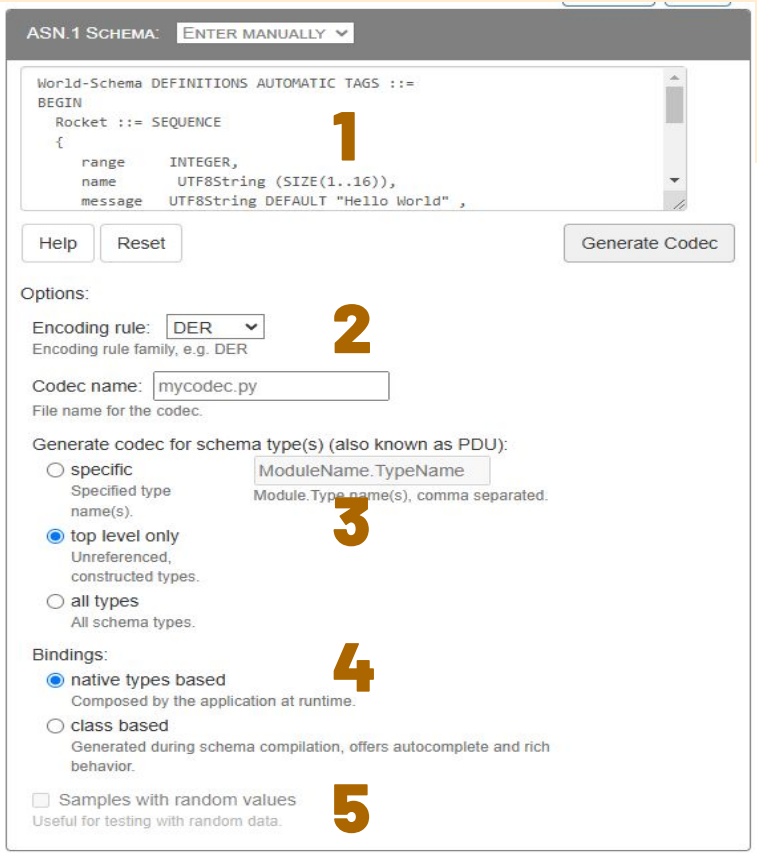
1. Get the schema
2. Specify the codec options
3. Generate and download a Python codec library
4. Start coding, get/set the data, call Encode/Decode/Validate.
5. Send/receive the binary stream over your connection.



Tips & Tricks

Pro tips that make you a **power-user**:

1. When your ASN.1 schema consists of more than one file, you can **upload multiple files** at once.
2. If you are developing a new protocol, consider OER as the most optimal (size vs speed). If you want the encodings to be **comparable in binary form**, use the canonical formats (COER, CUPER)
3. **Minimize generate code size** by specifying which schema types you want to serialize/communicate with.
4. Bindings represent the ASN.1 schema types in Python. "Native" bindings are dictionary based and require **no learning curve** to use, but are harder to manage for large/nested data. "Class" bindings get a separate Python class for each schema type, which adds some **type safety, type hints, auto-complete**.
5. For advanced testing you may need randomized data. This option allows you to **generate random values** for the types defined in your schema, so you do not have to write them manually.



Simple to code

My app

```
import Rocket

# set
value = json.loads(...)

# encode into OER binary stream
encoded = Rocket.World_Schema.Rocket.encode('OER', value)

# send/receive
. . .

# decode
decoded = Rocket.World_Schema.Rocket.decode('OER', encoded)

# validate
errors = Rocket.World_Schema.Rocket.validate(decoded)
```

encode

decode

validate

Generated codec API

Learn three functions

```
encode()
decode()
validate()
```

Manipulate the data (the bindings) as either **dictionaries**

```
dicObj = json.loads
('{"range": 0,...
```

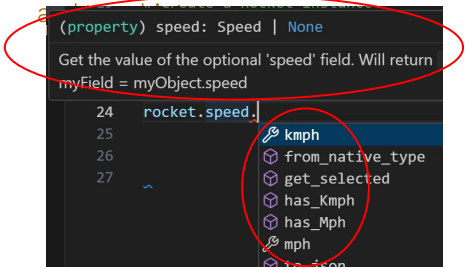
OR ***class-based objects***

```
msgObj = Rocket()
msgObj.range = 0
```

Schema Bindings to fit your needs

There are two alternatives for representing input/output values (aka bindings that bind Python values to the schema types):

- **native type bindings** - best for a quick start and simple data which can be manipulated via the Python dictionary objects.
- **class-based bindings** - best for complex data which can utilize the IDE Intellisense features like autocomplete



Native type bindings

```
import MyProto


# Set the native bindings
rocket = {
    "range": 350,
    "speed": { "kmph": 25000}
}

...
```

Class-based bindings

```
import MyProto
import bindings

# Create a Rocket instance
rocket = bindings.Rocket()
rocket.range = 350 #set the range

#add an optional field speed
rocket.speed = bindings.Rocket.Speed()
rocket.speed.kmph = 25000

...
```

Licensing

Suitable for your needs

	Individual licenses	Enterprise licenses
Dedicated customer support ● account Manager ● priority-support@oss.com	✗	✓
Multiple compiler versions ● early previews ● conformity and migration	✗	✓
Enterprise console ● team management ● team workspace	✗	✓
Bundled apps ● Complimentary with a compiler	✗	ASN.1 Playground, NAS Playground, ASN1Doc, Analyzer, ASN1Vcx

Learn more

- https://www.oss.com/asn1/resources/asn1-made-simple/why-asn1.html
  two major benefits of ASN.1 schema (compared to say a JSON schema) is **Expressivity and Authority**
- https://asn1.io/asn1-python-compiler/pyquickstart.html
  from a schema to running your code in a few simple steps explained in **Quick Start**
- https://asn1.io/doc/asn1pyweb/asn1-python-compiler-doc.html
  ASN.1 Python Compiler **Online documentation**
- https://www.oss.com/asn1/resources/asn1-made-simple/asn1-quick-reference.html
  ASN.1 **Quick Reference**

DIFFERENTIATORS

OSS offers more

ACCESSIBILITY

The online platform requires no installation and provides access from any browser on any platform.

SUPPORT

OSS offers comprehensive technical support with a long list of customers and decades of history.

EXPERTISE

OSS employs industry experts for all aspects of using ASN.1 in a right way

EASE OF USE

The web compiler is user-friendly and generates intuitive code that includes schema-specific samples and customizable options to fit your preferences.