<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Overview</td>
<td>3</td>
</tr>
<tr>
<td>2 Installation</td>
<td>5</td>
</tr>
<tr>
<td>3 Usage</td>
<td>7</td>
</tr>
<tr>
<td>4 Examples</td>
<td>11</td>
</tr>
<tr>
<td>5 Introduction to ASN.1</td>
<td>15</td>
</tr>
<tr>
<td>6 Reference</td>
<td>17</td>
</tr>
<tr>
<td>7 Contributing</td>
<td>21</td>
</tr>
<tr>
<td>8 Credits</td>
<td>23</td>
</tr>
<tr>
<td>9 Authors</td>
<td>25</td>
</tr>
<tr>
<td>10 License</td>
<td>27</td>
</tr>
<tr>
<td>11 Changelog</td>
<td>29</td>
</tr>
<tr>
<td>12 Indices and tables</td>
<td>31</td>
</tr>
<tr>
<td>Python Module Index</td>
<td>33</td>
</tr>
</tbody>
</table>
Python-ASN1 is a simple ASN.1 encoder and decoder for Python 2.6+ and 3.3+.

### 1.1 Features

- Support BER (parser) and DER (parser and generator) encoding
- 100% python, compatible with version 2.6, 2.7, 3.3 and higher
- Can be integrated by just including a file into your project

### 1.2 Dependencies

Python-ASN1 relies on Python-Future for Python 2 and 3 compatibility. To install Python-Future:

```bash
pip install future
```

### 1.3 How to install Python-asn1

Install from PyPi with the following:

```bash
pip install asn1
```

or download the repository from GitHub and install with the following:

```bash
python setup.py install
```

You can also simply include `asn1.py` into your project.
1.4 How to use Python-asn1

Note: You can find more detailed documentation on the Usage page.

1.4.1 Encoding

If you want to encode data and retrieve its DER-encoded representation, use code such as:

```python
import asn1

encoder = asn1.Encoder()
encoder.start()
encoder.write('1.2.3', asn1.ObjectIdentifier)
encoded_bytes = encoder.output()
```

1.4.2 Decoding

If you want to decode ASN.1 from DER or BER encoded bytes, use code such as:

```python
import asn1

decoder = asn1.Decoder()
decoder.start(encoded_bytes)
tag, value = decoder.read()
```

1.5 Documentation

The complete documentation is available on Read The Docs:

python-asn1.readthedocs.io

1.6 License

Python-ASN1 is free software that is made available under the MIT license. Consult the file LICENSE that is distributed together with this library for the exact licensing terms.

1.7 Copyright

The following people have contributed to Python-ASN1. Collectively they own the copyright of this software.

- Geert Jansen (geert@boskant.nl): original implementation.
- Sebastien Andrivet (sebastien@andrivet.com)
Installation

Install from PyPi with the following:

```
pip install asn1
```

Or download the repository from GitHub and install with the following:

```
python setup.py install
```

You can also simply include `asn1.py` into your project.
Note: You can find a complete example in Examples.

The Python-ASN1 API is exposed by a single Python module named \texttt{asn1}. The main interface is provided by the following classes:

- \texttt{Encoder}: Used to encode ASN.1.
- \texttt{Decoder}: Used to decode ASN.1.
- \texttt{Error}: Exception used to signal errors.

### 3.1 Type Mapping

The Python-ASN1 encoder and decoder make a difference between primitive and constructed data types. Primitive data types can be encoded and decoded directly with \texttt{read()} and \texttt{write()} methods. For these types, ASN.1 types are mapped directly to Python types and vice versa, as per the table below:

<table>
<thead>
<tr>
<th>ASN.1 type</th>
<th>Python type</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>bool</td>
<td>yes</td>
</tr>
<tr>
<td>Integer</td>
<td>int</td>
<td>yes</td>
</tr>
<tr>
<td>OctetString</td>
<td>bytes</td>
<td>yes</td>
</tr>
<tr>
<td>PrintableString</td>
<td>str</td>
<td>yes</td>
</tr>
<tr>
<td>Null</td>
<td>None</td>
<td>yes</td>
</tr>
<tr>
<td>ObjectIdentifier</td>
<td>bytes</td>
<td>no</td>
</tr>
<tr>
<td>Enumerated</td>
<td>int</td>
<td>no</td>
</tr>
</tbody>
</table>

The column \texttt{default} is relevant for encoding only. Because ASN.1 has more data types than Python, the situation arises that one Python type corresponds to multiple ASN.1 types. In this situation the to be encoded ASN.1 type cannot be determined from the Python type. The solution implemented in Python-ASN1 is that the most frequently used type will be the implicit default, and if another type is desired than that must be specified explicitly through the API.
For constructed types, no type mapping is done at all, even for types where such a mapping would be possible such as the ASN.1 type sequence of which could be mapped to a Python list. For such types a stack based approach is used instead. In this approach, the user needs to explicitly enter/leave the constructed type using the Encoder.enter() and Encoder.leave() methods of the encoder and the Decoder.enter() and Decoder.leave() methods of the decoder.

### 3.2 Encoding

If you want to encode data and retrieve its DER-encoded representation, use code such as:

```python
import asn1

encoder = asn1.Encoder()
encoder.start()
encoder.write('1.2.3', asn1.ObjectIdentifier)
encoded_bytes = encoder.output()
```

### 3.3 Decoding

If you want to decode ASN.1 from DER or BER encoded bytes, use code such as:

```python
import asn1

decoder = asn1.Decoder()
decoder.start(encoded_bytes)
tag, value = decoder.read()
```

### 3.4 Constants

A few constants are defined in the `asn1` module. The constants immediately below correspond to ASN.1 numbers. They can be used as the `nr` parameter of the Encoder.write() method, and are returned as the first part of a (nr, typ, cls) tuple as returned by Decoder.peek() and Decoder.read().

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value (hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>0x01</td>
</tr>
<tr>
<td>Integer</td>
<td>0x02</td>
</tr>
<tr>
<td>OctetString</td>
<td>0x04</td>
</tr>
<tr>
<td>Null</td>
<td>0x05</td>
</tr>
<tr>
<td>ObjectIdentifier</td>
<td>0x06</td>
</tr>
<tr>
<td>Enumerated</td>
<td>0x0a</td>
</tr>
<tr>
<td>Sequence</td>
<td>0x10</td>
</tr>
<tr>
<td>Set</td>
<td>0x11</td>
</tr>
</tbody>
</table>

The following constants define the two available encoding types (primitive and constructed) for ASN.1 data types. As above they can be used with the Encoder.write() and are returned by Decoder.peek() and Decoder.read().
Finally the constants below define the different ASN.1 classes. As above they can be used with the `Encoder.write()` and are returned by `Decoder.peek()` and `Decoder.read()`.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value (hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClassUniversal</td>
<td>0x00</td>
</tr>
<tr>
<td>ClassApplication</td>
<td>0x40</td>
</tr>
<tr>
<td>ClassContext</td>
<td>0x80</td>
</tr>
<tr>
<td>ClassPrivate</td>
<td>0xc0</td>
</tr>
</tbody>
</table>
4.1 Dump

This command line utility reads a X509.v3 certificate (in DER or PEM encoding), decodes it and outputs a textual representation. It more or less mimics openssl asn1parse:

```
$ python examples/dump.py examples/test.crt
```

```
[U] SEQUENCE
[U] SEQUENCE
[C] 0x0
    [U] INTEGER: 2
    [U] INTEGER: 5424
    [U] SEQUENCE
    [U] OBJECT: 1.2.840.113549.1.1.5
    [U] NULL: None
    [U] SEQUENCE
    [U] SET
        [U] SEQUENCE
            [U] OBJECT: 2.5.4.6
            [U] PRINTABLESTRING: --
        [U] SET
        [U] SEQUENCE
            [U] OBJECT: 2.5.4.8
            [U] PRINTABLESTRING: SomeState
        [U] SET
        [U] SEQUENCE
            [U] OBJECT: 2.5.4.7
            [U] PRINTABLESTRING: SomeCity
        [U] SET
        [U] SEQUENCE
            [U] OBJECT: 2.5.4.10
            [U] PRINTABLESTRING: SomeOrganization
        [U] SET
```

(continues on next page)
Python-ASN1, Release 2.2.0

(continued from previous page)

(U) SEQUENCE
  (U) OBJECT: 2.5.4.11
  (U) PRINTABLESTRING: SomeOrganizationalUnit
(U) SEQUENCE
  (U) OBJECT: 2.5.4.3
  (U) PRINTABLESTRING: localhost.localdomain
(U) SET
  (U) SEQUENCE
    (U) OBJECT: 1.2.840.113549.1.9.1
    (U) IA5STRING: root@localhost.localdomain
(U) SEQUENCE
  (U) UTCTIME: 080205092331Z
  (U) UTCTIME: 090204092331Z
(U) SET
  (U) SEQUENCE
    (U) OBJECT: 2.5.4.6
    (U) PRINTABLESTRING: --
(U) SET
  (U) SEQUENCE
    (U) OBJECT: 2.5.4.8
    (U) PRINTABLESTRING: SomeState
(U) SET
  (U) SEQUENCE
    (U) OBJECT: 2.5.4.7
    (U) PRINTABLESTRING: SomeCity
(U) SET
  (U) SEQUENCE
    (U) OBJECT: 2.5.4.10
    (U) PRINTABLESTRING: SomeOrganization
(U) SET
  (U) SEQUENCE
    (U) OBJECT: 2.5.4.11
    (U) PRINTABLESTRING: SomeOrganizationalUnit
(U) SET
  (U) SEQUENCE
    (U) OBJECT: 2.5.4.3
    (U) PRINTABLESTRING: localhost.localdomain
(U) SET
  (U) SEQUENCE
    (U) OBJECT: 1.2.840.113549.1.9.1
    (U) IA5STRING: root@localhost.localdomain
(U) SEQUENCE
  (U) SEQUENCE
    (U) OBJECT: 1.2.840.113549.1.1.1
    (U) NULL: None
    (U) BIT STRING: 0xb'0030818902818100D51...10610203010001'
(C) BIT STRING
(U) SEQUENCE
  (U) SEQUENCE
    (U) OBJECT: 2.5.29.14
    (U) OCTET STRING: 0xb'04140A4BFA8754177E30B4217156510FD291C3300236'
(U) SEQUENCE
  (U) OBJECT: 2.5.29.35
  (U) OCTET STRING: 0xb'3081DE80140A4...D61696E82021530'
(U) SEQUENCE

(continues on next page)
The main function is `pretty_print`:

```python
def pretty_print(input_stream, output_stream, indent=0):
    """Pretty print ASN.1 data."""
    while not input_stream.eof():
        tag = input_stream.peek()
        if tag.typ == asn1.TypePrimitive:
            tag, value = input_stream.read()
            output_stream.write(' ' * indent)
            output_stream.write('[{}]: {}
'.format(class_id_to_string(tag.cls),
                tag_id_to_string(tag.nr), value_to_string(tag.nr, value)))
        elif tag.typ == asn1.TypeConstructed:
            output_stream.write(' ' * indent)
            output_stream.write('[{}]
'.format(class_id_to_string(tag.cls),
                tag_id_to_string(tag.nr)))
            input_stream.enter()
            pretty_print(input_stream, output_stream, indent + 2)
            input_stream.leave()
```

This code:

- line 3: Loops until its reaches the end of the input stream (with `Decoder.eof()`).
- line 4: Looks (with `Decoder.peek()`) at the current tag.
- line 5: If the tag is primitive (`TypePrimitive`) ...
- line 6: ... the code reads (with `Decoder.read()`) the current tag.
- line 8: Then it displays its number and value (after some mapping to be more user-friendly).
- line 10: If the tag is constructed (`TypeConstructed`) ...
- line 12: ... the code displays its class and number.
- line 14: Then it enters inside the tag and ...
- line 15: ... calls itself recursively to decode the ASN.1 tags inside.
- line 16: Leaves the current constructed tag (with `Decoder.leave()`) to continue the decoding of its siblings.
ASN.1 is short for Abstract Syntax Notation One. It is a joint ISO/IEC and ITU-T standard for describing the syntax and encoding of data structures in a platform independent way. The syntax of data structures in ASN.1 is described by a syntax definition language. This language is used to compose ever more complex data types from a few simple ones such as integer and string. Often you will see this language in other standards that define an application on top of ASN.1 (for example LDAP). It is outside the scope of this introduction to go any deeper into the ASN.1 syntax definition language.

The encoding of data structures in ASN.1 is defined by so-called encoding rules. Not just one but many encoding rules exist, and occasionally new ones are added. Each of the encoding rules is powerful enough to encode any data type specified by the ASN.1 language. Because of the multiple encoding rules, it also means that it is not enough to know that a piece of data is ASN.1 to decode it: you also need to know the encoding rules with which it was encoded.

ASN.1 was first conceived in 1984. Some people say it is old, other say it is mature. Fact is that many Internet protocols are built on top of ASN.1. Some of the most important of these protocols are:

- **LDAP** (Lightweight Directory Access Protocol)
- **Kerberos** (network authentication protocol)
- **SNMP** (Simple Network Management Protocol)
- **X.509** (public key infrastructure, known from e.g. SSL certificates)

ASN.1 is not the only standard for defining the syntax and encoding of data structures. Other standards are Sun’s XDR (External Data Representation), and EBNF (Extended Backus-Naur Form). XDR is used primarily for ONC-RPC applications such as NFS (the Network File System). EBNF is perhaps the most used standard of them all: HTTP and XML (and thereby all its applications) are specified in EBNF.

## 5.1 ASN.1 Data Types or Tags

ASN.1 data types are either primitive data types such as integers and strings, or more complex ones built on them. Whatever the details of the data type, each of them has the following properties.

- **Its number**. The number is a numerical identifier that identifies the data type uniquely amongst others in its class. For example, ASN.1 defines that the type integer has the number 2.
• Its **class**. Four classes exist: **universal**, **application**, **context**, **private**. Each class defines the scope of the data type, i.e. how broad is the type available. Universal types are always available while the other ones are more restricted. All standard primitive types that are defined by ASN.1 are defined in the universal class.

• Its **encoding type** (or just **type**). Two types exist: **primitive** and **constructed**. Primitively types contain just one data value such as a number or a string, while constructed types contain a collection of other primitive or constructed types.

The three properties together (number, class, type) are also called the **tag** of a data type.

### 5.2 ASN.1 Encoding

As mentioned above, the encoding of data structures is defined by ASN.1 encoding rules. The most well-known encodings rules are:

• **BER** (Basic Encoding Rules)

• **DER** (Distinguished Encoding Rules)

The BER and DER encodings are very similar: BER is a looser format and sometimes allows a single value to be encoded in different ways. The DER is a subset of BER and defines in case multiple encodings are possible which one needs to be used. This means that a BER parser can read DER (because that is just a special form of BER), and a DER generator automatically generates valid BER (because all DER is also BER.)

**Python-ASN1** supports the DER and BER, and none of the other encoding rules. The vast majority of applications use these encoding rules, so at the moment no support for additional encoding rules is planned. When generating ASN.1, **Python-ASN1** will output DER, while for parsing it supports BER. This arrangement complies with the practise of being forgiving on input, and strict on output.

The DER and BER encodings work according the the **TLV** (Tag, Length, Value) principle. A stream of encoded ASN.1 data consists of a sequence of zero or more (tags, length, value) tuples, one after the other. These tuples are also called **records** in this manual, although this is not a term that is defined in the ASN.1 standard. For each data type, DER and BER define how to encode the tag (which we remember is a combination of its number, class and type), the length (in octets) of the data value, and the data value itself.

Because DER encoding results in a truly binary representation of the encoded data, a format has been devised for being able to send these in an encoding of printable characters so you can actually mail these things.

• **PEM** (Privacy Enhanced Mail) defined in RFC 1421. In essence PEM files are just base64 encoded versions of the DER encoded data. In order to distinguish from the outside what kind of data is inside the DER encoded string, a header and footer are present around the data.

### 5.3 References

• **ASN.1** ITU-T Study Group 17 - Languages for Telecommunication Systems

• **XDR** External Data Representation Standard

• **EBNF** ISO/IEC 14977: Information technology – Syntactic metalanguage – Extended BNF


6.1 asn1

This module provides ASN.1 encoder and decoder.

**class as1.Classes**

**class as1.Decoder**

ASN.1 decoder. Understands BER (and DER which is a subset).

**enter()**

This method enters the constructed type that is at the current decoding offset.

**Note:** It is an error to call `Decoder.enter()` if the to be decoded ASN.1 tag is not of a constructed type.

**Returns** None

**eof()**

Return True if we are at the end of input.

**Returns** bool – True if all input has been decoded, and False otherwise.

**leave()**

This method leaves the last constructed type that was `Decoder.enter()`-ed.

**Note:** It is an error to call `Decoder.leave()` if the current ASN.1 tag is not of a constructed type.

**Returns** None
peek()
This method returns the current ASN.1 tag (i.e. the tag that a subsequent Decoder.read() call would return) without updating the decoding offset. In case no more data is available from the input, this method returns None to signal end-of-file.

This method is useful if you don’t know whether the next tag will be a primitive or a constructed tag. Depending on the return value of peek, you would decide to either issue a Decoder.read() in case of a primitive type, or an Decoder.enter() in case of a constructed type.

Note: Because this method does not advance the current offset in the input, calling it multiple times in a row will return the same value for all calls.

Returns Tag – The current ASN.1 tag.
Raises Error

read(tagnr=None)
This method decodes one ASN.1 tag from the input and returns it as a (tag, value) tuple. tag is a 3-tuple (nr, typ, cls), while value is a Python object representing the ASN.1 value. The offset in the input is increased so that the next Decoder.read() call will return the next tag. In case no more data is available from the input, this method returns None to signal end-of-file.

Returns Tag, value – The current ASN.1 tag and its value.
Raises Error

start(data)
This method instructs the decoder to start decoding the ASN.1 input data, which must be a passed in as plain Python bytes. This method may be called at any time to start a new decoding job. If this method is called while currently decoding another input, that decoding context is discarded.

Note: It is not necessary to specify the encoding because the decoder assumes the input is in BER or DER format.

Parameters data (bytes) – ASN.1 input, in BER or DER format, to be decoded.
Returns None
Raises Error

class asn1.Encoder
ASN.1 encoder. Uses DER encoding.

enter(nr, cls=None)
This method starts the construction of a constructed type.

Parameters
• nr (int) – The desired ASN.1 type. Use Numbers enumeration.
• cls (int) – This optional parameter specifies the class of the constructed type. The default class to use is the universal class. Use Classes enumeration.

Returns None
Raises Error
leave()
This method completes the construction of a constructed type and writes the encoded representation to the output buffer.

output()
This method returns the encoded ASN.1 data as plain Python bytes. This method can be called multiple times, also during encoding. In the latter case the data that has been encoded so far is returned.

Note: It is an error to call this method if the encoder is still constructing a constructed type, i.e. if `Encoder.enter()` has been called more times that `Encoder.leave()`.

Returns bytes – The DER encoded ASN.1 data.
Raises Error

start()
This method instructs the encoder to start encoding a new ASN.1 output. This method may be called at any time to reset the encoder, and resets the current output (if any).

write(value, nr=None, typ=None, cls=None)
This method encodes one ASN.1 tag and writes it to the output buffer.

Note: Normally, value will be the only parameter to this method. In this case Python-ASN1 will autodetect the correct ASN.1 type from the type of value, and will output the encoded value based on this type.

Parameters

• value (any) – The value of the ASN.1 tag to write. Python-ASN1 will try to autodetect the correct ASN.1 type from the type of value.

• nr (int) – If the desired ASN.1 type cannot be autodetected or is autodetected wrongly, the nr parameter can be provided to specify the ASN.1 type to be used. Use Numbers enumeration.

• typ (int) – This optional parameter can be used to write constructed types to the output by setting it to indicate the constructed encoding type. In this case, value must already be valid ASN.1 encoded data as plain Python bytes. This is not normally how constructed types should be encoded though, see `Encoder.enter()` and `Encoder.leave()` for the recommended way of doing this. Use Types enumeration.

• cls (int) – This parameter can be used to override the class of the value. The default class is the universal class. Use Classes enumeration.

Returns None
 Raises Error

exception asn1.Error
ASN.11 encoding or decoding error.

class asn1.Numbers

class asn1.Tag(nr, typ, cls)
A named tuple to represent ASN.1 tags as returned by `Decoder.peek()` and `Decoder.read()`.

6.1. asn1
```
cls
    Alias for field number 2

nr
    Alias for field number 0

typ
    Alias for field number 1

class asn1.Types
```
Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

## 7.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

## 7.2 Documentation improvements

Python-ASN1 could always use more documentation, whether as part of the official Python-ASN1 docs, in docstrings, or even on the web in blog posts, articles, and such.

## 7.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/andrivet/python-asn1/pulls.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

Contributing
7.4 Development

To set up Python-ASN1 for local development:

1. Fork python-asn1 (look for the “Fork” button).
2. Clone your fork locally:
   
   ```
   git clone git@github.com:your_name_here/python-asn1.git
   ```
3. Create a branch for local development:
   
   ```
   git checkout -b name-of-your-bugfix-or-feature
   ```
   
   Now you can make your changes locally.
4. When you’re done making changes, run all the checks, doc builder and spell checker with tox one command:
   
   ```
   tox
   ```
5. Commit your changes and push your branch to GitHub:
   
   ```
   git add .
   git commit -m "Your detailed description of your changes."
   git push origin name-of-your-bugfix-or-feature
   ```
6. Submit a pull request through the GitHub website.

7.4.1 Pull Request Guidelines

If you need some code review or feedback while you’re developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run tox)\(^1\).
2. Update documentation when there’s new API, functionality etc.
3. Add a note to CHANGELOG.rst about the changes.
4. Add yourself to AUTHORS.rst.

7.4.2 Tips

To run a subset of tests:

```
 tox -e envname -- py.test -k test_myfeature
```

To run all the test environments in parallel (you need to pip install detox):

```
detox
```
The following projects have provided inspiration for Python-ASN1:

- **pyasn1** This is another ASN1 encoder/decoder for Python.
- **Samba** In particular **libads** for the stack based approach for building constructed types.
The following people have contributed to Python-ASN1. Collectively they own the copyright of this software.

- Geert Jansen <geert@boskant.nl>
- Sebastien Andrivet <sebastien@andrivet.com>
- Finian Blackett <finian.blackett@gmail.com>
Copyright (c) 2007-2016 the Python-ASN1 authors.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
2.1.1 (2017-10-30)

- Fix a bug (#9): two’s complement corner case with values such as -32769. Add new test cases to test them.

11.1 2.1.0 (2016-12-18)

- Add more documentation
- Use (simulated) enumerations
- Add Python 2.6 in automated checks and tests
- Add type hints (for static checking) and fix some code

11.2 2.0.0 (2016-12-16)

- First public release by Sebastien Andrivet
- Support both python 2 and 3 (with Python-Future)
- All strings are now in unicode
- Add more ASN.1 tags (like PrintableString)
- Fix errors in the example (dump.py)
- Code reorganization

11.3 0.9 (2011-05-18)

- Initial public release by Geert Jansen
CHAPTER 12

Indices and tables

• genindex
• search
Python Module Index

a
asn1, 17
Index

A
asn1 (module), 17

C
Classes (class in asn1), 17
cls (asn1.Tag attribute), 19

D
Decoder (class in asn1), 17

E
Encoder (class in asn1), 18
enter() (asn1.Decoder method), 17
enter() (asn1.Encoder method), 18
eof() (asn1.Decoder method), 17
Error, 19

L
leave() (asn1.Decoder method), 17
leave() (asn1.Encoder method), 18

N
nr (asn1.Tag attribute), 20
Numbers (class in asn1), 19

O
output() (asn1.Encoder method), 19

P
peek() (asn1.Decoder method), 17

R
read() (asn1.Decoder method), 18

S
start() (asn1.Decoder method), 18
start() (asn1.Encoder method), 19

T
Tag (class in asn1), 19
typ (asn1.Tag attribute), 20
Types (class in asn1), 20

W
write() (asn1.Encoder method), 19