14.5. `plistlib` — Generate and parse Mac OS X `.plist` files

Source code: Lib/plistlib.py

This module provides an interface for reading and writing the "property list" files used mainly by Mac OS X and supports both binary and XML plist files.

The property list (`.plist`) file format is a simple serialization supporting basic object types, like dictionaries, lists, numbers and strings. Usually the top level object is a dictionary.

To write out and to parse a plist file, use the `dump()` and `load()` functions.

To work with plist data in bytes objects, use `dumps()` and `loads()`.

Values can be strings, integers, floats, booleans, tuples, lists, dictionaries (but only with string keys), `Data`, `bytes`, `bytesarray` or `datetime.datetime` objects.

Changed in version 3.4: New API, old API deprecated. Support for binary format plists added.

See also:

PList manual page
Apple's documentation of the file format.

This module defines the following functions:

`plistlib.`**load**(*fp*, *, *fmt=None*, *use_builtin_types=True*, *dict_type=dict*)
Read a plist file. *fp* should be a readable and binary file object. Return the unpacked root object (which usually is a dictionary).

The *fmt* is the format of the file and the following values are valid:

- `None`: Autodetect the file format
- `FMT_XML`: XML file format
- `FMT_BINARY`: Binary plist format

If *use_builtin_types* is true (the default) binary data will be returned as instances of `bytes`, otherwise it is returned as instances of `Data`.

The *dict_type* is the type used for dictionaries that are read from the plist file. The exact structure of the plist can be recovered by using `collections.OrderedDict` (although the order of keys shouldn't be important in plist files).

XML data for the `FMT_XML` format is parsed using the Expat parser from `xml.parsers.expat` – see its documentation for possible exceptions on ill-formed XML. Unknown elements will simply be ignored by the plist parser.

The parser for the binary format raises `InvalidFileException` when the file cannot be parsed.

New in version 3.4.

`plistlib.`**`loads`**(*data*, ***, *fmt=None*, *use_builtin_types=True*, *dict_type=dict*)

Load a plist from a bytes object. See `load()` for an explanation of the keyword arguments.

New in version 3.4.

`plistlib.`**`dump`**(*value*, *fp*, ***, *fmt=FMT_XML*, *sort_keys=True*, *skipkeys=False*)

Write *value* to a plist file. *Fp* should be a writable, binary file object.

The *fmt* argument specifies the format of the plist file and can be one of the following values:

- `FMT_XML`: XML formatted plist file
- `FMT_BINARY`: Binary formatted plist file

When *sort_keys* is true (the default) the keys for dictionaries will be written to the plist in sorted order, otherwise they will be written in the iteration order of the dictionary.

When *skipkeys* is false (the default) the function raises `TypeError` when a key of a dictionary is not a string, otherwise such keys are skipped.

A `TypeError` will be raised if the object is of an unsupported type or a container that contains objects of unsupported types.

An `OverflowError` will be raised for integer values that cannot be represented in (binary) plist files.

New in version 3.4.

`plistlib.`**`dumps`**(*value*, ***, *fmt=FMT_XML*, *sort_keys=True*, *skipkeys=False*)

Return *value* as a plist-formatted bytes object. See the documentation for `dump()` for an explanation of the keyword arguments of this function.

New in version 3.4.

The following functions are deprecated:

`plistlib.`**`readPlist`**(*pathOrFile*)
Read a plist file. *pathOrFile* may be either a file name or a (readable and binary) file object. Returns the unpacked root object (which usually is a dictionary).

This function calls `load()` to do the actual work, see the documentation of `that function` for an explanation of the keyword arguments.

> **Note:** Dict values in the result have a `__getattr__` method that defers to `__getitem_`. This means that you can use attribute access to access items of these dictionaries.

Deprecated since version 3.4: Use `load()` instead.

`plistlib.`**`writePlist`**(*rootObject*, *pathOrFile*)
Write *rootObject* to an XML plist file. *pathOrFile* may be either a file name or a (writable and binary) file object

Deprecated since version 3.4: Use `dump()` instead.

`plistlib.`**`readPlistFromBytes`**(*data*)
Read a plist data from a bytes object. Return the root object.

See `load()` for a description of the keyword arguments.

> **Note:** Dict values in the result have a `__getattr__` method that defers to `__getitem_`. This means that you can use attribute access to access items of these dictionaries.

Deprecated since version 3.4: Use `loads()` instead.

`plistlib.`**`writePlistToBytes`**(*rootObject*)
Return *rootObject* as an XML plist-formatted bytes object.

Deprecated since version 3.4: Use `dumps()` instead.

The following classes are available:

`Dict([dict]):`
Return an extended mapping object with the same value as dictionary *dict*.

This class is a subclass of `dict` where attribute access can be used to access items. That is, `aDict.key` is the same as `aDict['key']` for getting, setting and deleting items in the mapping.

Deprecated since version 3.0.

class `plistlib.`**`Data`**(*data*)

Return a "data" wrapper object around the bytes object *data*. This is used in functions converting from/to plists to represent the `<data>` type available in plists.

It has one attribute, `data`, that can be used to retrieve the Python bytes object stored in it.

Deprecated since version 3.4: Use a `bytes` object instead.

The following constants are available:

`plistlib.`**`FMT_XML`**

The XML format for plist files.

New in version 3.4.

`plistlib.`**`FMT_BINARY`**

The binary format for plist files

New in version 3.4.

14.5.1. Examples

Generating a plist:

```
pl = dict(
    aString = "Doodah",
    aList = ["A", "B", 12, 32.1, [1, 2, 3]],
    aFloat = 0.1,
    anInt = 728,
    aDict = dict(
        anotherString = "<hello & hi there!>",
        aThirdString = "M\xe4ssig, Ma\xdf",
        aTrueValue = True,
        aFalseValue = False,
    ),
    someData = b"<binary gunk>",
    someMoreData = b"<lots of binary gunk>" * 10,
    aDate = datetime.datetime.fromtimestamp(time.mktime(time.gmtime())
)
```

```
with open(fileName, 'wb') as fp:
    dump(pl, fp)
```

Parsing a plist:

```
with open(fileName, 'rb') as fp:
    pl = load(fp)
print(pl["aKey"])
```