javaproperties Documentation

Release 0.1.0

John T. Wodder II

2016 Oct 03

Contents

Contents:

`javaproperties` provides support for reading & writing Java `.properties` files (both the "plain" format and XML) with a simple API based on the `json` module — though, for recovering Java addicts, it also includes a `Properties` class intended to match the behavior of Java 8's `java.net.Properties` as much as is Pythonically possible.

Note: Throughout the following, "text string" means a Unicode character string — `unicode` in Python 2, `str` in Python 3.

CHAPTER 1

Reading & Writing `.properties` Files

javaproperties.**dump**(*props*, *fp*, *separator=u'='*, *comments=None*, *timestamp=True*, *sort_keys=False*)

Write a series of key-value pairs to a file in `.properties` format.

Parameters

- **props** – A mapping or iterable of `(key, value)` pairs to write to `fp`. All keys and values in `props` must be text strings.
- **fp** – A file-like object to write the values of `props` to. It must have been opened as a text file with a Latin-1-compatible encoding.
- **separator** (*text string*) – the string to use for separating keys & values. Only `" "`, `"="`, and `":"` (possibly with added whitespace) should ever be used as the separator.
- **comments** (text string or `None`) – if non-`None`, `comments` will be written to `fp` as a comment before any other content
- **timestamp** (`None`, `bool`, number, or `datetime.datetime`) – If neither `None` nor `False`, a timestamp in the form of `Mon Sep 12 14:00:54 EDT 2016` is written as a comment to `fp` after `comments` (if any) and before the key-value pairs. If `timestamp` is `True`, the current date & time is used. If it is a number, it is converted from seconds since the epoch to local time. If it is a `datetime.datetime` object, its value is used directly, with naïve objects assumed to be in the local timezone.
- **sort_keys** (*bool*) – if true, the elements of `props` are sorted lexicographically by key in the output

Returns `None`

javaproperties.**dumps**(*props*, *separator=u'='*, *comments=None*, *timestamp=True*, *sort_keys=False*)

Convert a series of key-value pairs to a text string in `.properties` format.

Parameters

- **props** – A mapping or iterable of `(key, value)` pairs to serialize. All keys and values in `props` must be text strings.
- **separator** (*text string*) – the string to use for separating keys & values. Only `" "`, `"="`, and `":"` (possibly with added whitespace) should ever be used as the separator.
- **comments** (text string or `None`) – if non-`None`, `comments` will be output as a comment before any other content
- **timestamp** (`None`, `bool`, number, or `datetime.datetime`) – If neither `None` nor `False`, a timestamp in the form of `Mon Sep 12 14:00:54 EDT 2016` is output as a comment after `comments` (if any) and before the key-value pairs. If `timestamp` is

`True`, the current date & time is used. If it is a number, it is converted from seconds since the epoch to local time. If it is a `datetime.datetime` object, its value is used directly, with naïve objects assumed to be in the local timezone.

- **sort_keys** (*bool*) – if true, the elements of `props` are sorted lexicographically by key in the output

Return type text string

`javaproperties.`**load**(*fp*, *object_pairs_hook=<type 'dict'>*)

Parse the contents of the `readline`-supporting file-like object `fp` as a `.properties` file and return a `dict` of the key-value pairs.

`fp` may be either a text or binary filehandle, with or without universal newlines enabled. If it is a binary filehandle, its contents are decoded as Latin-1.

By default, the key-value pairs extracted from `fp` are combined into a `dict` with later occurrences of a key overriding previous occurrences of the same key. To change this behavior, pass a callable as the `object_pairs_hook` argument; it will be called with one argument, a generator of `(key, value)` pairs representing the key-value entries in `fp` (including duplicates) in order of occurrence. *load* will then return the value returned by `object_pairs_hook`.

Parameters

- **fp** (*file-like object*) – the file from which to read the `.properties` document
- **object_pairs_hook** (*callable*) – class or function for combining the key-value pairs

Return type `dict` of text strings or the return value of `object_pairs_hook`

`javaproperties.`**loads**(*s*, *object_pairs_hook=<type 'dict'>*)

Parse the contents of the string `s` as a `.properties` file and return a `dict` of the key-value pairs.

`s` may be either a text string or bytes string. If it is a bytes string, its contents are decoded as Latin-1.

By default, the key-value pairs extracted from `s` are combined into a `dict` with later occurrences of a key overriding previous occurrences of the same key. To change this behavior, pass a callable as the `object_pairs_hook` argument; it will be called with one argument, a generator of `(key, value)` pairs representing the key-value entries in `s` (including duplicates) in order of occurrence. *loads* will then return the value returned by `object_pairs_hook`.

Parameters

- **s** (*string*) – the string from which to read the `.properties` document
- **object_pairs_hook** (*callable*) – class or function for combining the key-value pairs

Return type `dict` of text strings or the return value of `object_pairs_hook`

CHAPTER 2

Reading & Writing `.properties` Files as XML

javaproperties.**dump_xml**(*props*, *fp*, *comment=None*, *encoding=u'UTF-8'*, *sort_keys=False*)

Write a series `props` of key-value pairs to a binary filehandle `fp` in the format of an XML properties file. The file will include both an XML declaration and a doctype declaration.

Parameters

- **props** – A mapping or iterable of `(key, value)` pairs to write to `fp`. All keys and values in `props` must be text strings.
- **fp** (*binary file-like object*) – a file-like object to write the values of `props` to
- **comment** (text string or `None`) – if non-`None`, `comment` will be output as a `<comment>` element before the `<entry>` elements
- **encoding** (*string*) – the name of the encoding to use for the XML document (also included in the XML declaration)
- **sort_keys** (*bool*) – if true, the elements of `props` are sorted lexicographically by key in the output

Returns `None`

javaproperties.**dumps_xml**(*props*, *comment=None*, *sort_keys=False*)

Convert a series `props` of key-value pairs to a text string containing an XML properties document. The document will include a doctype declaration but not an XML declaration.

Parameters

- **props** – A mapping or iterable of `(key, value)` pairs to serialize. All keys and values in `props` must be text strings.
- **comment** (text string or `None`) – if non-`None`, `comment` will be output as a `<comment>` element before the `<entry>` elements
- **sort_keys** (*bool*) – if true, the elements of `props` are sorted lexicographically by key in the output

Return type text string

javaproperties.**load_xml**(*fp*, *object_pairs_hook=<type 'dict'>*)

Parse the contents of the file-like object `fp` as an XML properties file and return a `dict` of the key-value pairs.

Beyond basic XML well-formedness, *load_xml* only checks that the root element is named `properties` and that all of its `entry` children have `key` attributes; no further validation is performed.

By default, the key-value pairs extracted from `fp` are combined into a `dict` with later occurrences of a key overriding previous occurrences of the same key. To change this behavior, pass a callable as the

object_pairs_hook argument; it will be called with one argument, a generator of (key, value) pairs representing the key-value entries in fp (including duplicates) in order of occurrence. *load_xml* will then return the value returned by object_pairs_hook.

Parameters

- **fp** (*file-like object*) – the file from which to read the XML properties document
- **object_pairs_hook** (*callable*) – class or function for combining the key-value pairs

Return type dict or the return value of object_pairs_hook

Raises **ValueError** – if the root of the XML tree is not a <properties> tag or an <entry> element is missing a key attribute

javaproperties.**loads_xml**(*s*, *object_pairs_hook=<type 'dict'>*)

Parse the contents of the string s as an XML properties document and return a dict of the key-value pairs.

Beyond basic XML well-formedness, *loads_xml* only checks that the root element is named properties and that all of its entry children have key attributes; no further validation is performed.

By default, the key-value pairs extracted from s are combined into a dict with later occurrences of a key overriding previous occurrences of the same key. To change this behavior, pass a callable as the object_pairs_hook argument; it will be called with one argument, a generator of (key, value) pairs representing the key-value entries in s (including duplicates) in order of occurrence. *loads_xml* will then return the value returned by object_pairs_hook.

Parameters

- **s** (*string*) – the string from which to read the XML properties document
- **object_pairs_hook** (*callable*) – class or function for combining the key-value pairs

Return type dict or the return value of object_pairs_hook

Raises **ValueError** – if the root of the XML tree is not a <properties> tag or an <entry> element is missing a key attribute

CHAPTER 3

`Properties` Class

class `javaproperties.`**`Properties`**(*data=None*, *defaults=None*)

A port of Java 8's `java.net.Properties` that tries to match its behavior as much as is Pythonically possible. *`Properties`* behaves like a normal `MutableMapping` class (i.e., you can do `props[key] = value` and so forth), except that it may only be used to store strings (`str` and `unicode` in Python 2; just `str` in Python 3). Attempts to use a non-string object as a key or value will produce a `TypeError`.

Parameters

- **`data`** (mapping or `None`) – A mapping or iterable of `(key, value)` pairs with which to initialize the *`Properties`* object. All keys and values in `data` must be text strings.
- **`defaults`** (*`Properties`* or `None`) – a set of default properties that will be used as fallback for *`getProperty`*

`defaults` = **None**

A *`Properties`* subobject used as fallback for *`getProperty`*. Only *`getProperty`*, *`propertyNames`*, and *`stringPropertyNames`* use this attribute; all other methods (including the standard mapping methods) ignore it.

`getProperty`(*key*, *defaultValue=None*)

Fetch the value associated with the key `key` in the *`Properties`* object. If the key is not present, *`defaults`* is checked, and then *its* *`defaults`*, etc., until either a value for `key` is found or the next *`defaults`* is `None`, in which case `defaultValue` is returned.

Parameters

- **`key`** (*`text string`*) – the key to look up the value of
- **`defaultValue`** – the value to return if `key` is not found in the *`Properties`* object

Return type text string (if `key` was found)

Raises **`TypeError`** – if `key` is not a string

`load`(*inStream*)

Update the *`Properties`* object with the entries in a `.properties` file or file-like object.

`inStream` may be either a text or binary filehandle, with or without universal newlines enabled. If it is a binary filehandle, its contents are decoded as Latin-1.

Parameters **`inStream`** (*`file-like object`*) – the file from which to read the `.properties` document

Returns `None`

`loadFromXML`(*inStream*)

Update the *`Properties`* object with the entries in the XML properties file `inStream`.

Beyond basic XML well-formedness, *loadFromXML* only checks that the root element is named `properties` and that all of its `entry` children have `key` attributes; no further validation is performed.

Parameters **inStream** (*file-like object*) – the file from which to read the XML properties document

Returns None

Raises **ValueError** – if the root of the XML tree is not a `<properties>` tag or an `<entry>` element is missing a `key` attribute

propertyNames()

Returns a generator of all distinct keys in the *Properties* object and its *defaults* (and its *defaults*'s *defaults*, etc.) in unspecified order

Return type generator of text strings

setProperty(*key*, *value*)

Equivalent to `self[key] = value`

store(*out*, *comments=None*)

Write the *Properties* object's entries (in unspecified order) in `.properties` format to `out`, including the current timestamp.

Parameters

- **out** – A file-like object to write the properties to. It must have been opened as a text file with a Latin-1-compatible encoding.
- **comments** (text string or None) – If non-None, `comments` will be written to `out` as a comment before any other content

Returns None

storeToXML(*out*, *comment=None*, *encoding='UTF-8'*)

Write the *Properties* object's entries (in unspecified order) in XML properties format to `out`.

Parameters

- **out** (*binary file-like object*) – a file-like object to write the properties to
- **comment** (text string or None) – if non-None, `comment` will be output as a `<comment>` element before the `<entry>` elements
- **encoding** (*string*) – the name of the encoding to use for the XML document (also included in the XML declaration)

Returns None

stringPropertyNames()

Returns a `set` of all keys in the *Properties* object and its *defaults* (and its *defaults*'s *defaults*, etc.)

Return type `set` of text strings

CHAPTER 4

Low-Level Utilities

javaproperties.**escape**(*field*)

Escape a string so that it can be safely used as either a key or value in a `.properties` file. All non-ASCII characters, all nonprintable or space characters, and the characters `\ # ! = :` are all escaped using either `\uXXXX` escapes (after converting non-BMP characters to surrogate pairs) or the single-character escapes recognized by *unescape*.

Parameters **field** (*text string*) – the string to escape

Return type text string

javaproperties.**join_key_value**(*key*, *value*, *separator=u'='*)

Join a key and value together into a single line suitable for adding to a `.properties` file.

Parameters

- **key** (*text string*) – the key
- **value** (*text string*) – the value
- **separator** (*text string*) – the string to use for separating the key & value. Only `" "`, `"="`, and `":"` (possibly with added whitespace) should ever be used as the separator.

Return type text string

javaproperties.**parse**(*fp*)

Parse the contents of the `readline`-supporting file-like object `fp` as a `.properties` file and return a generator of `(key, value, original_lines)` triples, including duplicate keys and including comments & blank lines (which have their `key` and `value` fields set to `None`). This is the only way to extract comments from a `.properties` file.

`fp` may be either a text or binary filehandle, with or without universal newlines enabled. If it is a binary filehandle, its contents are decoded as Latin-1.

Parameters **fp** (*file-like object*) – the file from which to read the `.properties` document

Return type generator of triples of text strings

javaproperties.**to_comment**(*comment*)

Convert a string to a `.properties` file comment. All non-Latin-1 characters in the string are escaped using `\uXXXX` escapes (after converting non-BMP characters to surrogate pairs), a `#` is prepended to the string, and a `#` is inserted after any line breaks in the string not already followed by a `#` or `!`.

Parameters **comment** (*text string*) – the string to convert to a comment

Return type text string

javaproperties.**unescape**(*field*)

Decode escape sequences in a `.properties` key or value. The following escape sequences are recognized:

```
\t \n \f \r \uXXXX \\
```

If a backslash is followed by any other character, the backslash is dropped.

In addition, any valid UTF-16 surrogate pairs in the string after escape-decoding are further decoded into the non-BMP characters they represent. (Invalid & isolated surrogate code points are left as-is.)

Parameters **field** (*text string*) – the string to decode

Return type text string

CHAPTER 5

Command-Line Interface

javaproperties includes two command-line programs for converting `.properties` files to & from the much more widely-supported JSON format. Currently, only "plain"/non-XML properties files are supported.

5.1 json2properties

```
python -m javaproperties.fromjson [infile [outfile]]
# or, if the javaproperties package was properly installed:
json2properties [infile [outfile]]
```

Convert a JSON file `infile` to a Latin-1 `.properties` file and write the results to `outfile`. If not specified, `infile` and `outfile` default to `sys.stdin` and `sys.stdout`, respectively.

The JSON document must be an object with scalar (i.e., string, numeric, boolean, and/or null) values; anything else will result in an error.

Output is sorted by key, and numeric, boolean, & null values are output using their JSON representations; e.g., the input:

```
{
    "yes": true,
    "no": "false",
    "nothing": null
}
```

becomes:

```
#Mon Sep 26 18:57:44 UTC 2016
no=false
nothing=null
yes=true
```

5.2 properties2json

```
python -m javaproperties.tojson [infile [outfile]]
# or, if the javaproperties package was properly installed:
properties2json [infile [outfile]]
```

Convert a Latin-1 `.properties` file `infile` to a JSON object and write the results to `outfile`. If not specified, `infile` and `outfile` default to `sys.stdin` and `sys.stdout`, respectively.

CHAPTER 6

Indices and tables

- genindex
- search

j

D

E

G

J

L

P

S

T

U