pyvalem

Release 2.0

Christian Hill

Mar 23, 2020

CONTENTS:

CHAPTER

ONE

INTRODUCTION TO PYVALEM

PyValem is a Python package for parsing, validating, manipulating and interpreting the chemical formulas, quantum states and labels of atoms, ions and small molecules.

Species and states are specfied as strings using a simple and flexible syntax, and may be compared, output in different formats and manipulated using a variety of predefined Python methods.

1.1 Installation

TODO

1.2 Examples

The basic (state-less) chemical formula class is `Formula`. A `Formula` object can be created by passing its constructor a valid string. This object contains attributes for producing its plain text, HTML and LaTeX representations, and for calculating its molar mass:

```
In [1]: from pyvalem.formula import Formula

In [2]: f = Formula('C2H5OH')

In [3]: print(f)
C2H5OH

In [4]: print(f.html)
C<sub>2</sub>H<sub>5</sub>OH

In [5]: print(f.latex)
$\mathrm{C}_{2}\mathrm{H}_{5}\mathrm{O}\mathrm{H}$

In [6]: print(f.rmm)     # g.mol-1
46.069
```

Note that there is no underscore character (_) before between the element symbol and its stoichiometry. Isotopes are specified with the mass number placed before the element symbol, with both surrounded by parentheses. Do not use a caret (^) to indicate a superscript:

```
In [7]: f = Formula('(14C)')

In [8]: print(f.html)
```

(continues on next page)

(continued from previous page)

```
<sup>14</sup>C

In [9]: print(f.rmm)
14.0032419884

In [10]: f = Formula('H2(18O)')

In [11]: print(f.rmm)
20.015159612799998
```

For isotopically-pure compounds, the mass returned is the atomic mass.

Charges are specified as `+n` or `-n`, where `n` may be omitted if it is 1. Do not use a caret (`^`) to indicate a superscript:

```
In [12]: f = Formula('H3O+')
In [13]: print(f.charge)
1

In [14]: print(f.html)
H<sub>3</sub>O<sup>+</sup>

In [15]: f = Formula('Co(H2O)6+2')
In [16]: print(f.charge)
2

In [17]: print(f.html)
Co(H<sub>2</sub>O)<sub>6</sub><sup>2+</sup>
```

"Stateful" species are formulas which consist of a valid `Formula` string, followed by whitespace, followed by a semicolon-delimited sequence of valid quantum state or label specifications. Stateful species know which states they possess and can render these states in different ways. For example:

```
In [18]: from pyvalem.stateful_species import StatefulSpecies
In [19]: ss1 = StatefulSpecies('Ne+ 1s2.2s2.2p5; 2P_1/2')
In [20]: ss1.states
Out[21]: [1s2.2s2.2p5, 2P_1/2]

In [22]: ss1.states[1].__class__
Out[22]: pyvalem.atomic_term_symbol.AtomicTermSymbol

In [23]: ss1.html
Out[23]: 'Ne<sup>+</sup> 1s<sup>2</sup>2s<sup>2</sup>2p<sup>5</sup>; <sup>2</sup>P
↪<sub>1/2</sub>'
```

This HTML renders as:

Ne^{+} $1s^2 2s^2 2p^5$; $^{2}P_{1/2}$

Another example:

```
In [24]: ss2 = StatefulSpecies('(52Cr)(1H) 1σ2.2σ1.1δ2.1π2; 6Σ+; v=0; J=2')
In [25]: ss2.html
<sup>52</sup>Cr<sup>1</sup>H 1σ<sup>2</sup>.2σ<sup>1</sup>.1δ<sup>2</sup>.1π<sup>2</sup>; <sup>6</sup>Σ<sup>+</sup>; v=0; J=2
```

which produces:

^{52}Cr^{1}H $1\sigma^2.2\sigma^1.1\delta^2.1\pi^2$; $^{6}\Sigma^{+}$; $v = 0$; $J = 2$

The syntax for writing different types of quantum state are described in later pages of this documentation.

CHAPTER

TWO

FORMULA

A `Formula` object instance represents the chemical formula of an atom, isotope, ion, molecule, molecular ion, or certain sorts of other particle. The `Formula` class has methods for producing representations of the formula in HTML, LaTeX and plain text.

`Formula` objects are not supposed to be unique (different `Formula` objects can represent the same formula); nor is their syntax or this library designed to be applied to large or complex molecules. PyValem is intended as a lightweight, easy-to-use library with an expressive syntax for representing many common small atoms and molecules and their isotopes, states and reactions.

Furthermore, whilst some validation functionality is built into the library, PyValem does not attempt to verify that a provided formula is chemically plausible. In particular, it knows nothing about valence or oxidation state.

2.1 Instantiation

A `Formula` object may be instantiated by passing a valid string, conforming to the following grammar:

- Single atoms, with atomic weights given by a default natural isotopic abundance are specfied with their element symbol, e.g. `H`, `Be`, `Fr`.
- Isotopes are specified in parentheses (round brackets) with the isotope mass number preceding the element symbol, e.g. `(12C)`, `(35Cl)`, `(235U)`. Note that no caret (`^`) is used to indicate a superscript.
- Charged species are specified with the charge following the formula in the format `+n` or `-n`, where `n` may be omitted if it is 1. Do not use a caret (`^`) to indicate a superscript. For example, `He+`, `C+2`, `W-`, `(79Br)-2`.
- Molecular formulas are written as a sequence of element symbols (which may be repeated for clarity over the structure), with their stoichiometries specified as an integer following the symbol. No underscore (_) character is used. For example, `H2O`, `(1H)2(16O)`, `C2H6OH`, `CH3CH2OH`, `NH+`, `CO3+2`.
- Moieties within formula can be bracketed for clarity, for example `CH3C(CH3)2CH3`.
- A limited number of formula prefixes are supported, for example `L-CH3CH(NH2)CO2H`, `cis-CH3CHCHCH3`, `ortho-C6H4(CH3)2`
- **There are some special species:**
  - `e-` is the electron;
  - `e+` is the positron;
  - `M` is a generic third-body with no specific identity (and no defined mass or charge);
  - `hv` (or `hv`) is the photon.

2.2 Output as HTML, LaTeX and slugs

The `Formula` attributes `html` and `latex` return strings representing the formula in HTML and LaTeX respectively. The attribute `slug` returns a URL-safe slug which uniquely identifies the formula's plain text string. For example:

```
In [1]: from pyvalem.formula import Formula

In [2]: f = Formula('')
In [3]: print(f.formula)       # or simply print(f)
Co(H2O)6+2

In [4]: print(f.html)
Co(H<sub>2</sub>O)<sub>6</sub><sup>2+</sup>

In [5]: print(f.latex)
\mathrm{Co}(\mathrm{H}_{2}\mathrm{O})_{6}^{2+}

In [6]: print(f.slug)
Co-_l_H2O_r_6_p2
```

The HTML and LaTeX representations render as:

$\mathrm{Co}(\mathrm{H_2O})_{6}^{2+}$

2.3 Formula Attributes

`Formula` objects can count atoms, calculate masses and record the total species charge:

```
In [7]: f = Formula('CO3-2')
In [8]: print(f.natoms)
4
In [9]: print(f.rmm)
60.008

In [10]: print(f.charge)
-2

In [11]: lys = Formula('(NH3+)(CH2)4CH(NH2)CO2-')
In [12]: print(lys.natoms)
24

In [13]: print(lys.rmm)     # relative molecular mass
146.19

In [14]: print(lys.charge)
0
```

This last example is the Lysine zwitterion,

$(\mathrm{NH_3^+})(\mathrm{CH_2})_4\mathrm{CH}(\mathrm{NH_2})\mathrm{CO_2^-}$

The same applies to isotopes and isotopically-pure molecules, in which case the exact mass is held by the `mass` attribute:

```
In [15]: f = formula('(1H)(35Cl)+')
In [16]: print(f.mass)
35.9766777262
```

The stoichiometric formula can be output either in order of increasing atomic number (the default) or in alphabetical order:

```
In [17]: print(lys.stoichiometric_formula())
H14C6N2O2

In [18]: print(lys.stoichiometric_formula('alphabetical'))
C6H14N2O2
```

CHAPTER

THREE

STATES

PyValem recognises several different kinds of quantum states and labels and is able to deduce the state type, parse quantum numbers and symmetries from provided strings, and do some validation.

States of different types can be instantiated directly from their class or associated with a species formula as a `StatefulSpecies` object.

3.1 GenericExcitedState

A "generic excited state" is simply an unspecified excitation of the species above its ground state: it may represent excitation of a single species to an unspecified quantum level or the general excitation of an ensemble of such species to a range of levels under equilibrium or non-equilibrium conditions.

A `GenericExcitedState` object is instantiated as a string of 1-4 asterisks or as a number preceding a single asterisk. For example:

```
In [1]: from pyvalem.generic_excited_state import GenericExcitedState
In [2]: x0 = GenericExcitedState('*')
In [3]: x1 = GenericExcitedState('***')
In [4]: x2 = GenericExcitedState('5*')
```

3.2 AtomicConfiguration

The electronic configuration of atoms, ions and isotopes is represented by the `AtomicConfiguration` class. The string representation provided to instantiate an `AtomicConfiguration` object is a sequence of orbital specifiers, separated by a full stop (period, `.`). An atomic orbital is represented by its principal quantum number, `n`, the azimuthal quantum number, `l` written as the letter `s`, `p`, `d`, …, and the an occupancy (number of electrons, `1` or `2`). For example:

```
In [1]: from pyvalem.atomic_configuration import AtomicConfiguration
In [2]: c0 = AtomicConfiguration('1s2')
In [3]: c1 = AtomicConfiguration('1s2.2s2.2p6')
In [4]: c2 = AtomicConfiguration('1s2.2s2.2p6.3s2.3d10')
```

The closed-shell part of an atomic configuration can be specified by providing the corresponding noble gas element symbol in square brackets:

```
In [5]: c2 =  AtomicConfiguration('[Ne].3s2.3d10')
```

The HTML representation of the atomic configuration is returned by the `html` attribute:

```
In [5]: print(c2)
[Ne]3s<sup>2</sup>3d<sup>10</sup>
```

which renders as:

$[\mathrm{Ne}]3s^2 3d^{10}$

If the same atomic orbital appears more than once or an orbital is specified as being occupied by more than the number of electrons allowed by the Pauli exclusion principle, an `AtomicConfigurationError` is raised:

```
In [6]: c3 = AtomicConfiguration('[Ar].3s2')
...
AtomicConfigurationError: Repeated subshell in [Ar].3s2

In [7]: c3 = AtomicConfiguration('1s2.2s2.2p7')
...
AtomicConfigurationError: Too many electrons in atomic orbital: 2p7
```

3.3 AtomicTermSymbol

The `AtomicTermSymbol` class represents an atomic electronic term symbol in the LS-coupling (Russell–Saunders coupling) notation. The total electronic spin quantum number, S, is specifed by a multiplicity, 2S+1, followed by the total electronic orbital angular momentum quantum number, L, as a letter, `S`, `P`, `D`, … for L = 0, 1, 2, …; there may optionally follow a parity label, `o` (for odd-parity states), and a total (spin-orbit coupled) electronic angular momentum quantum number, J after an underscore character (_). Half (odd)-integers are specfied as fractions: `1/2`, `3/2`, `5/2`, etc.

Examples:

```
In [1]: from pyvalem.atomic_term_symbol import AtomicTermSymbol
In [2]: a0 = AtomicTermSymbol('3P_1/2')
In [3]: a1 = AtomicTermSymbol('4D')
In [4]: a2 = AtomicTermSymbol('2Po_1/2')
```

`AtomicTermSymbol` objects know about their quantum numbers, where defined:

```
In [5]: a0.S, a0.L, a0.J
Out[5]: (1.0, 1, 0.5)

In [6]: a1.S, a1.L, a1.J
Out[6]: (1.5, 2, None)

In [7]: a2.S, a2.L, a2.J, a2.parity
Out[7]: (0.5, 1, 0.5, 'o')
```

As with other states, the `html` attribute returns the HTML representation:

```
In [8]: print(a2.html)
<sup>2</sup>P<sup>o</sup><sub>1/2</sub>
```

which renders as:

$^2P^o_{1/2}$

J must satisfy the “triangle rule” (|L-S| ≤ J ≤ L+S); if this test fails, an `AtomicTermSymbolError` is raised:

```
In [9]: a3 = AtomicTermSymbol('3P_3')
...
AtomicTermSymbolError: Invalid atomic term symbol: 3P_3 |L-S| <= J <= L+S must be␣
↪satisfied.
```

3.4 DiatomicMolecularConfiguration

The electronic configuration of diatomic molecules, molecular ions and their isotopologues is represented by the `DiatomicMolecularConfiguration` class. This class is instantiated with a string consisting of a sequence of numbered molecular orbitals, separated by a full stop (period, `.`). Each orbital is denoted with a counting number (which increments for each orbital of the same symmetry), an orbital symmetry label, and an electron occupancy number.

The valid symmetry labels are the lower case greek letters, `σ`, `π`, and `δ`, with an optional letter, `u` or `g` denoting the inversion symmetry for centrosymmetric molecules. For convenience, the identifiers `sigma`, `pi`, `delta` may be used as identifiers instead of the greek labels.

Examples:

```
In [1]: from pyvalem.diatomic_molecular_configuration import␣
↪DiatomicMolecularConfiguration
In [2]: c1 = DiatomicMolecularConfiguration('1σ')
In [3]: c2 = DiatomicMolecularConfiguration('1σu2')
In [3]: c3 = DiatomicMolecularConfiguration('1piu4.1pig3')
In [4]: c4 = DiatomicMolecularConfiguration('1σg2.1σu2.2σg2.2σu2.1πu4.3σg2')
```

An HTML representation of the configuration is held in the `html` attribute:

```
In [5]: print(c3.html)
1π<sub>u</sub><sup>4</sup>.1π<sub>g</sub><sup>3</sup>

In [6]: print(c4.html)
1σ<sub>g</sub><sup>2</sup>.1σ<sub>u</sub><sup>2</sup>.2σ<sub>g</sub><sup>2</sup>.2σ
↪<sub>u</sub><sup>2</sup>.1π<sub>u</sub><sup>4</sup>.3σ<sub>g</sub><sup>2</sup>
```

These render as:

$1\pi_u^4 1\pi_g^3$ and $1\sigma_g^2 1\sigma_u^2 2\sigma_g^2 2\sigma_u^2 1\pi_u^4 3\sigma_g^2$

An attempt to fill an orbital with too many electrons raises a `DiatomicMolecularConfigurationError`, as does repeating an orbital:

```
In [7]: c5 = DiatomicMolecularConfiguration('1σ2.2σ2.3σ3')
...
DiatomicMolecularConfigurationError: Only two electrons allowed in σ orbital, but␣
↪received 3

In [8]: c6 = DiatomicMolecularConfiguration('1σ2.2σ2.2σ1')
...
DiatomicMolecularConfigurationError: Repeated orbitals in 1σ2.2σ2.2σ1
```

3.5 MolecularTermSymbol

The `MolecularTermSymbol` class represents a molecular electronic term symbol. It is instantiated with a string providing the spin multiplicity (2S+1), electronic orbital angular momentum symmetry label and (optionally) the quantum number Ω in the Hund's case (a) formulation.

Symmetry labels are specified as an appropriate letter combination representing the irreducible representation of the electronic orbital wavefunction. For molecules with a centre of symmetry, the `u`/`g` label comes last, preceded by any +/− label denoting the reflection symmetry of the wavefunction. Do not use a caret (^) or underscore (_) character to indicate superscripts or subscripts. For linear molecules, the irrep symbols Σ, Π, Δ and Φ may be replaced with `SIGMA`, `PI`, `DELTA`, and `PHI`.

If a term symbol has a label, this is given before the term symbol itself, which should be enclosed with parentheses.

Examples:

```
In [1]: from pyvalem.molecular_term_symbol import MolecularTermSymbol
In [2]: c1 = MolecularTermSymbol('1Σ-')
In [3]: c2 = MolecularTermSymbol('1SIGMA-')
In [4]: c3 = MolecularTermSymbol('3Σ+g')
In [5]: c4 = MolecularTermSymbol('3SIGMA+g')
In [7]: c5 = MolecularTermSymbol('b(4Π_-3/2)')
In [8]: c6 = MolecularTermSymbol("A'(1A1g_0)")
In [9]: c7 = MolecularTermSymbol('A(1A")')
```

The `html` attribute returns the HTML representation of the molecular term symbol. The examples above are represented by:

$^{1}\Sigma^{-}$
$^{1}\Sigma^{-}$
$^{3}\Sigma_{g}^{+}$
$^{3}\Sigma_{g}^{+}$
$b(^{4}\Pi_{-3/2})$
$A'(^{1}A_{1g0})$
$A(^{1}A'')$

`MolecularTermSymbol` objects know about their total spin angular momentum quantum number, S, the electronic orbital angular momentum wavefunction irrep, the projection of the total angular momentum along the symmetry axis, Ω, and the term label. For example:

```
In [10]: c6 = MolecularTermSymbol('X(4Πu_-3/2)')
In [11]: c6.S, c6.term_label, c6.irrep, c6.Omega
Out[11]: (1.5, 'X', 'Πu', -1.5)
```

3.6 VibrationalState

Molecular vibrational states of diatomic and polyatomic molecules are represented by the `VibrationalState` class. For diatomic molecules, this class is instantiated with a string of the form `v=n` for `n=0,1,2,...`. Polyatomic vibrational states are initialised with a string giving the number of quanta in each labelled normal mode. Unspecified vibrational excitation is denoted `v=*`. For example:

```
In [1]: from pyvalem.vibrational_state import VibrationalState
In [2]: v0 = VibrationalState('v=0')
In [3]: v1 = VibrationalState('v=*')
```

(continues on next page)

(continued from previous page)

```
In [4]: v2 = VibrationalState('3v2+v3')
In [5]: v3 = VibrationalState('v1+v2')
In [6]: v4 = VibrationalState('2v1 + 3v4')
In [7]: v5 = VibrationalState('2v1+1v2+v3')
```

The attribute `html` holds an HTML representation of the vibrational state:

```
In [8]: print(v5.html)
2v<sub>1</sub> + v<sub>2</sub> + v<sub>3</sub>
```

This renders as:

$2\nu_1 + \nu_2 + \nu_3$

3.7 RotationalState

A single class, `RotationalState`, represents the total rotational angular momentum (excluding nuclear spin) quantum number, J, for both atoms and molecules. It is instantiated with a string, `J=n`, where `n` is an integer or half integer (expressed as a fraction or in decimal notation). Three different levels of unspecified rotation excitation may alternatively be specified by providing `n` as one to three asterisks. The parsed value of J is available as an attribute with that name.

Examples:

```
In [1]: from pyvalem.rotational_state import RotationalState
In [2]: r0 = RotationalState('J=0')
In [3]: r1 = RotationalState('J=3/2')
In [4]: r2 = RotationalState('J=1.5')
In [5]: r3 = RotationalState('J=*')
In [6]: print(r1.J)
1.5
```

3.8 KeyValuePair

The `KeyValuePair` class represents an arbitrary quantum number or symmetry provided as a (key, value) pair. It is instantiated with a string of the form `key=value`. For example:

```
In [1]: from pyvalem.key_value_pair import KeyValuePair
In [2]: kv1 = KeyValuePair('n=1')
In [3]: kv2 = KeyValuePair('C = 45a#')
```

CHAPTER

FOUR

STATEFUL SPECIES

A *stateful species* is a species identified by a chemical formula which is associated with one or more quantum states of labels, and can be represented by an instance of the `StatefulSpecies` class.

A `StatefulSpecies` object can be instantiated by providing a string consisting of the formula (which can be parsed into a `Formula` object), followed by whitespace and a semicolon delimited sequence of strings that can be parsed by one of the `State` classes described previously. The state type is deduced from the format of the string.

Examples:

```
In [1]: from pyvalem.stateful_species import StatefulSpecies
In [2]: ss1 = StatefulSpecies('HCl v=2;J=0')
In [3]: ss2 = StatefulSpecies('NCO v2+3v1;J=10.5;a(2Σ-g)')
In [4]: ss3 = StatefulSpecies('Ar+ 1s2.2s2.2p5; 2P_3/2')
In [5]: ss4 = StatefulSpecies('CrH 1sigma2.2sigma1.1pi4.3sigma1; 6SIGMA+')
```

An HTML representation is accessible through the `html` attribute:

```
In [6]: print(ss4.html)
CrH 1σ<sup>2</sup>.2σ<sup>1</sup>.1π<sup>4</sup>.3σ<sup>1</sup>; <sup>6</sup>Σ<sup>+</
↪sup>
```

This example renders as:

CrH $1\sigma^2 2\sigma^1 1\pi^4 3\sigma^1;\ ^6\Sigma^+$

The `Formula` object and a list of the parse `State` objects are returned by the `formula` and `states` attributes:

```
In [7]: print(ss4.formula)
CrH

In [8]: print(ss4.states)
[1σ2.2σ1.1π4.3σ1, 6Σ+]
```

Two `StatefulSpecies` are considered the same (equal) if they have equal `Formula` objects and matching states:

```
In [9]: ss5 = StatefulSpecies('CO2 v2+3v1; J=2')
In [10]: ss6 = StatefulSpecies('CO2 J=2; v2+3v1')
In [11]: ss5 == ss6
Out[11]: True
```

Some `State` types cannot be repeated within a `StatefulSpecies` object (e.g. it doesn't make sense for a molecule to have two rotational quantum numbers, J). This is checked for:

```
In [12]: StatefulSpecies('N2 J=0;J=1')
...
StatefulSpeciesError: Multiple states of type RotationalState specified for N2 J=0;J=1

In [13]: StatefulSpecies('Li 1s2.2s1; 1s2.2p1')
...
StatefulSpeciesError: Multiple states of type AtomicConfiguration specified for Li␣
↪1s2.2s1;1s2.2p1
```

CHAPTER

FIVE

REACTION

A `Reaction` object is a collection of reactants and a collection of products, each of which is represented by a `StatefulSpecies` (though an individual species need not be associated with a specified state). It is instantiated using a string consisting of reactant and product `StatefulSpecies`, each separated by a plus sign surrounded by whitespace: `' + '`. Reactants and products are separated by any of the following strings: `' → ', ' = ', ' ⇌ ', ' -> ', ' <-> ', ' <=> '`. Examples:

```
In [1]: from pyvalem.reaction import Reaction
In [2]: r1 = Reaction('CO + O2 → CO2 + O')
In [3]: r2 = Reaction('CO v=1 + O2 J=2;X(3SIGMA-g) → CO2 + O')
In [4]: r3 = Reaction('HCl + hv -> H+ + Cl + e-')
In [5]: r4 = Reaction('C6H5OH + 7O2 → 6CO2 + 3H2O')
```

An HTML representation of the reaction is returned by the `html` attribute:

```
In [6] print(r2.html)
CO v=1 + O<sub>2</sub> J=2; X(<sup>3</sup>Σ<sup>-</sup><sub>g</sub>) → CO<sub>2</sub>␣
↪+ O
In [7] print(r4.html)
C<sub>6</sub>H<sub>5</sub>OH + 7O<sub>2</sub> → 6CO<sub>2</sub> + 3H<sub>2</sub>O
```

These examples render as:

$\mathrm{CO}\ v = 1 + \mathrm{O_2}\ J = 2;\ X(^3\Sigma_g^-) \rightarrow \mathrm{CO_2} + \mathrm{O}$
$\mathrm{C_6H_5OH} + 7\mathrm{O_2} \rightarrow 6\mathrm{CO_2} + 3\mathrm{H_2O}$

`Reaction` objects are validated to ensure that the charge and stoichiometry are conserved:

```
In [8]: Reaction('BeH+ + I2 ⇌ BeI + HI')
...
ReactionChargeError: Charge not preserved for reaction: BeH+ + I2 ⇌ BeI + HI

In [9]: Reaction('BeH + I2 ⇌ BeI')
...
ReactionStoichiometryError: Stoichiometry not preserved for reaction: BeH + I2 ⇌ BeI
```

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search