Spec for the Element enumeration

Public Enum Element

* Should enumerate each of the elements from Hydrogen (1) to Lawrencium (103)
* Should provide access to metadata about each element via extension methods
* Integer value of enumeration should be the atomic number

Extension methods

Function Configuration() as ElectronicConfiguration

* Returns the ground state configuration as an object instance.

Function AtomicWeight() as Double

* Returns the atomic weight of the element as a Double.
* Optional method.

Function Symbol() as String

* Returns the symbol for the element as a String.

Spec for the ElectronicConfiguration class

Public Structure ElectronicConfiguration

Encapsulates an electronic configuration (1s2 2s2 2p6 etc) and exposes methods to retrieve the occupancy of each orbital.

Implementation – store as a collection or list, keyed by the primary quantum number N concatenated with L quantum number, delimited. Ex: 2p6 🡪 key = “2p”, value = 6.0. OR have an enumeration of orbitals, ex: 1s = 0, 2s = 1, 2p = 2, 3s = 3, etc., and store occupation as corresponding array value. The former method may be superior from a parsing perspective. Can enumerate L value as s = 0, p = 1, d = 2, etc.

Should be able to parse input as full list of orbitals, and also referenced to noble gas configuration, ex: “[Kr] 4d10 5p2”

Should also handle input format string omitting occupation if = 1, ex: “1s” = “1s1”

Sub New(configuration as String)

* Parses the input string
* Throws an InvalidOperationException if the input string is not properly formatted.

Function Occupancy(n as Integer, l as Integer) as Double

* Returns the number of electrons in the orbital defined by quantum number n and angular momentum l.

Overloads Function Occupancy() as Double

* Returns the total number of electrons in the configuration. Can be fractional.
* Empty configuration has ElectronCount = 0.

Function ToString() as String

* Returns the configuration formatted as a space-delimited string. Ex: “1s2 2s2 2p5”
* Empty configuration returns an empty string.

Spec for the Atom class

On creation, the Atom class solves for the potential and wave functions using the method of Herman and Skillman (use strategy pattern here to try different methods)

For starters the strategy is to make the object solve at creation and remain read-only thereafter. After initial coding I’ll do a performance analysis to see how feasible it would be to make inputs read/write and trigger recalculations (would help make the class usable for muffin-tin atom)

Public Class Atom

Private mConfiguration as ElectronicConfiguration

Sub New(Z as Element)

Sub New(Z as Element, configuration as ElectronicConfiguration)

Readonly Property Mesh() as IRadialMesh

Readonly Property Ionicity() as Double

Readonly Property MuffinTinRadius() as Double

Readonly Property Pnl(n as Integer, l as Integer) as RadialWaveFunction

Readonly Property Potential() as RadialPotential

Readonly Property Energy(n as Integer, l as Integer) as Double

Do I use an AtomFactory and keep the solvers/algorithms outside? That makes the Atom class just a data structure. Since most of the data in the Atom is independent of algorithm, that kind of makes sense. I can swap out factories to change algorithms.

Function GetAtom(Z as element, configuration as ElectronicConfiguration, strategy as AtomSolverStrategy)

Construction:

* From Z, get the ground state configuration
* Set up the mesh
* Initialize the HermanSkillman Solver (input parameters)
* Construct the starting trial potential
* Do the self-consistent solve
  + For each orbital (multithread for other orbitals), IRadialEquationSolver
    - Get trial energy
    - Do outward/inward integration solve
      * Use generalized Numerov (mesh independent)
      * Do outward integral and count nodes.
      * Do inward integral
    - Compare logarithmic derivatives
    - Loop to new trial energy
  + Construct new trial potential
  + Check self-consistency conditions and exit if satisfied

IRadialEquationSolver

Interface for different methods of solving the radial Schrödinger equation. Given an input potential, n and l, solves for the energy eigenvalue and radial wavefunction. Supports asynchronous operation on multiple threads.

Sub New(potential as RadialPotential, mesh as IRadialMesh,

Function Solve()

Class RadialWaveFunction

Encapsulates a (normalized) radial wave function.

Sub New(E as double, n as integer, l as integer, Pnl() as Double(), mesh as IRadialMesh)

Function Pnl(index as Integer) as Double

Function Rnl(index as Integer) as Double

Function R(index as Integer) as Double

* Pass through to Mesh.X(index)

Readonly Property N as integer

* Returns the primary quantum number n. Must be greater than or equal to zero.

Readonly Property L as integer

* Returns the angular momentum quantum number l. Must be greater than or equal to zero.

Readonly Property E as Double

* Returns the energy eigenvalue of the state. Is less than zero for bound states.

Function Sigma(index as Integer) as Double

* Returns the spherically-averaged radial charge density.

Function Rho(index as Integer) as Double

* Returns the radial charge density.