What is the httk, and what can it do for YOU?

Rickard Armiento

February 5, 2015

Overview

The High-Throughput ToolKit (httk)

- A toolkit for preparing and running calculations, analyzing the result, store them (in a global or personalized database).
- The primary focus is automatization: run with as little human intervention as possible.
- *Necessary* for large datasets, but also *convinient* for smaller projects!
- Intended to expand beyond atomistic calculations, but that is our focus for now.

Overview

- Python library for:
  - Handling crystal structures.
  - Prepare calculations to be run.
  - Storage, retrival, search and analysis of data in database.
- Scripts/programs for:
  - Handling large sets of computer runs.
  - System for scripting advanced multi-stage runs to be run on clusters with limited walltime.
  - Managing ongoing runs across many supercomputer clusters.

Overview

- Why not use/extend similar libraries instead (ASE, pymatgen, etc.)?
  - Different core design choices that make local database interaction as easy as possible; allows storage, powerful searching, and analysis of data stored in regular python objects in your own local database.
  - Core classes, e.g., crystal structures and lattices, preserve decimal numbers on exact form (helps a lot especially with crystal geometry and database comparisons...)
  - Can be installed and used without non-standard libraries; not even numpy, scipy are required. Only calls on other libraries and software when needed. (And then, httk even helps you to load those libraries from odd locations, e.g., to avoid old system-wide versions.)
  - When you need features in ASE, pytmatgen, etc., httk translates to and from their formats.

Examples: Structures

Trivial to load a cif file, or poscar, etc.:

```
import httk

struct = httk.load("example.cif")

print("Formula:", struct.formula)
print("Volume", float(struct.uc_volume))
print("Assignments", struct.uc_symbols)
print("Counts:", struct.uc_counts)
print("Coords", struct.uc_reduced_coords)
```

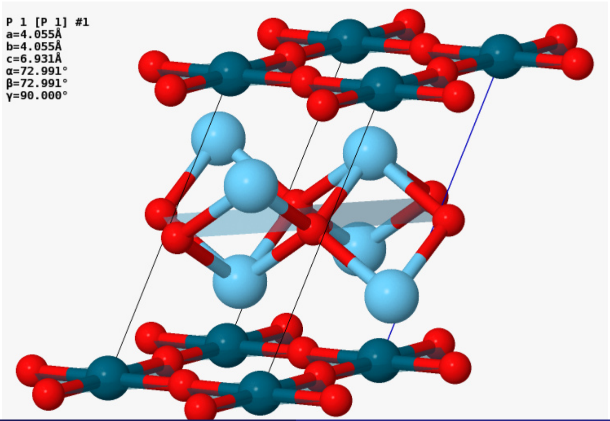
Output:

```
('Formula:', 'BO2Tl')
('Volume', 509.24213999999984)
('Assignments', ['B', 'O', 'Tl'])
('Counts:', [8, 16, 8])
('Coords', FracVector(((1350, 4550, 4250), ..., ,10000)))
```

Examples: Visualization

Easy visualization in, e.g., jmol

```
import httk
import httk.atomistic.vis

struct = httk.load("POSCAR")
struct.vis.show()
```



Examples: Structure

Of course one can also create and modify structures directly in code:

```
from httk.atomistic import Structure

cell = [[1.0, 0.0, 0.0],
        [0.0, 1.0, 0.0],
        [0.0, 0.0, 1.0]]

coordgroups = [[
                  [0.5, 0.5, 0.5]
               ],[
                  [0.0, 0.0, 0.0]
               ],[
                  [0.5, 0.0, 0.0],[0.0, 0.5, 0.0],[0.0, 0.0, 0.5]
              ]]

assignments = ['Pb','Ti','O']
volume=62.79
struct = Structure.create(uc_cell=cell,
            uc_reduced_coordgroups=coordgroups,
            assignments=assignments,
            uc_volume=volume)
```

Examples: Structure

Find symmetry information (calls out to ISOTROPY software):

```
import httk

struct = httk.load("POSCAR")

print("Formula:", struct.formula)
hall_symbol = struct.hall_symbol
spacegroup_number = struct.spacegroup_number
print("Spacegroup info:",hall_symbol, spacegroup_number)
```

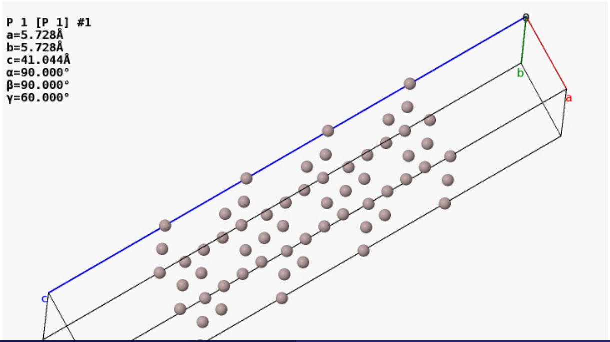
Output:

```
('Formula:', 'O3PbTi')
('Spacegroup info', '-P 4 2 3', 221)
```

Examples: External libraries

Easy interaction with other useful libraries and software, e.g., ASE:
(Presently: ASE, aflow, cif2cell, isotropy, jmol, pymatgen, platon, gulp)

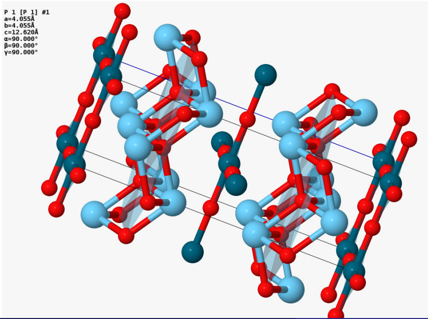
```
import httk
import httk.atomistic.vis
import httk.external.ase_glue
import ase

slab = ase.lattice.surface.fcc111('Al', size=(2,2,10), vacuum=10.0)
struct2 = Structure.ase.from_Atoms(slab)
struct.vis.show()
```

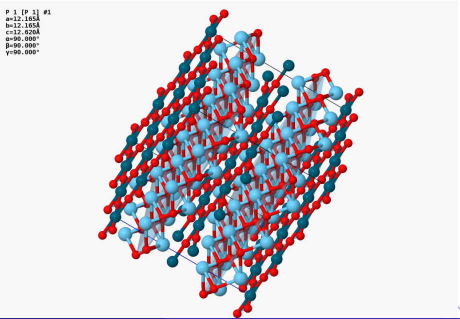


Examples: Supercells

Build supercells, find orthogonal and cubic ones:

```python
import httk

struct = httk.load("POSCAR2")
supercell1 = struct.build_supercell([[2,0,0],[0,2,0],[0,0,1]])
struct.vis.show()
supercell2 = struct.build_orthogonal_supercell(tolerance=20)
struct.vis.show()
supercell3 = struct.build_cubic_supercell(tolerance=20)
struct.vis.show()
```



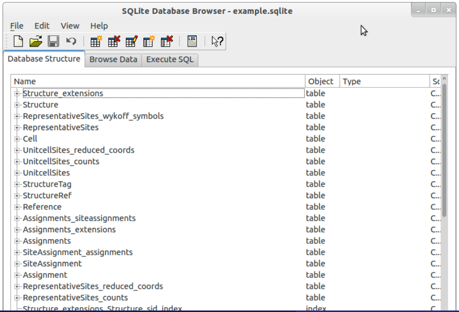


Examples: Database

Store data in a local relational database (sqlite):

```
import httk
import httk.db

backend = httk.db.backend.Sqlite('example.sqlite')
store = httk.db.store.SqlStore(backend)
struct = httk.load('example.cif')
store.save(struct)
```

```
user@computer:> sqlitebrowser example.sqlite
```



Name	Object	Type	Sc
Structure_extensions	table		C...
Structure	table		C...
RepresentativeSites_wykoff_symbols	table		C...
RepresentativeSites	table		C...
Cell	table		C...
UnitcellSites_reduced_coords	table		C...
UnitcellSites_counts	table		C...
UnitcellSites	table		C...
StructureTag	table		C...
StructureRef	table		C...
Reference	table		C...
Assignments_siteassignments	table		C...
Assignments_extensions	table		C...
Assignments	table		C...
SiteAssignment_assignments	table		C...
SiteAssignment	table		C...
Assignment	table		C...
RepresentativeSites_reduced_coords	table		C...
RepresentativeSites_counts	table		C...
Structure_extensions_Structure_sid_index	index		C

Examples: Database

Search in your local database:

```
import httk
import httk.db

backend = httk.db.backend.Sqlite('database.sqlite')
store = httk.db.store.SqlStore(backend)

search = store.searcher()
search_struct = search.variable(Structure)
search.add( search_struct.uc_nbr_atoms < 40 )

search.output(search_struct,'structure')

for header, match in search:
    structure = match[0]
    print("Found:",struct.formula)
```

Output:

```
('Found:','ZnO2')
```

Examples: Computations

Generate a simple vasp calculation:

```
import httk
import httk.iface.vasp_if

poscarspath="/path/to/your/poscars/POT_GGA_PAW_PBE/"

struct = httk.load("example.cif")

httk.iface.vasp_if.prepare_single_run("Run", struct, template='
    example', poscarspath=poscarspath)
```

```
user@computer:> cd Run
user@computer:> vasp
```

Output:

```
 running on     1 nodes
 distr:  one band on     1 nodes,     1 groups
 vasp.5.2.12 11Nov11 complex

 POSCAR found type information on POSCAR  Ti N
 POSCAR found :  2 types and       2 ions

 ------------------------------------------------------------------------
```

Examples: Computations

Generate a big batch of computations:

```
import httk
import httk.d

backend = httk.db.backend.Sqlite('../tutorial.sqlite')
store = httk.db.store.SqlStore(backend)

search = store.searcher()
search_struct = search.variable(Structure)
search.add( search_struct.uc_nbr_atoms < 40 )

search.output (search_struct,'structure')

for match in search:
    structure = match[0][0]
    print("Found:",struct.formula)
```

Examples: Computations

Execute set of computations:

```
import httk
import httk.db

backend = httk.db.backend.Sqlite('../tutorial.sqlite')
store = httk.db.store.SqlStore(backend)

search = store.searcher()
search_struct = search.variable(Structure)
search.add( search_struct.uc_nbr_atoms < 40 )

search.output (search_struct,'structure')

for match in search:
    structure = match[0][0]
    print("Found:",struct.formula)
```

Examples: Computations

Read results back into database

```
import httk
import httk.db

backend = httk.db.backend.Sqlite('../tutorial.sqlite')
store = httk.db.store.SqlStore(backend)

search = store.searcher()
search_struct = search.variable(Structure)
search.add( search_struct.uc_nbr_atoms < 40 )

search.output (search_struct,'structure')

for match in search:
    structure = match[0][0]
    print("Found:",struct.formula)
```

Examples: Computations

Search in resulting data

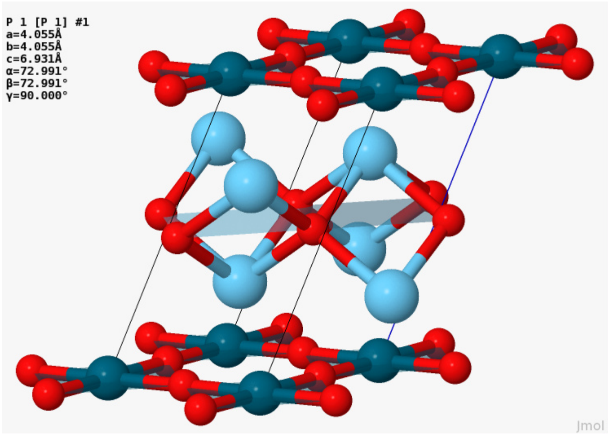
```
import httk
import httk.db

backend = httk.db.backend.Sqlite('../tutorial.sqlite')
store = httk.db.store.SqlStore(backend)

search = store.searcher()
search_struct = search.variable(Structure)
search.add( search_struct.uc_nbr_atoms < 40 )

search.output (search_struct,'structure')

for match in search:
    structure = match[0][0]
    print("Found:",struct.formula)
```

Examples: Global database

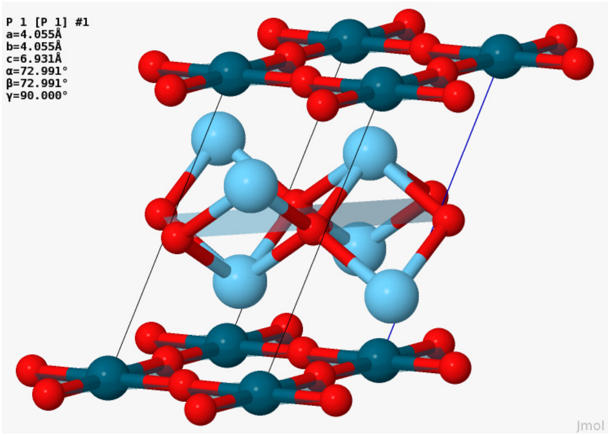
Submit to central database

```
import httk
import httk.db

backend = httk.db.backend.Sqlite('../tutorial.sqlite')
store = httk.db.store.SqlStore(backend)

search = store.searcher()
search_struct = search.variable(Structure)
search.add( search_struct.uc_nbr_atoms < 40 )

search.output (search_struct,'structure')

for match in search:
    structure = match[0][0]
    print("Found:",struct.formula)
```

The Open Materials Database



The *Closed* Materials Database



Central database

Operate directly on data in central database:
(Still to be implemented)

```
import httk
import httk.db

store = httk.db.open_materials_database_store

search = store.searcher()
search_struct = search.variable(Structure)
search.add( search_struct.uc_nbr_atoms < 40 )

search.output (search_struct,'structure')

for match in search:
    structure = match[0][0]
    print("Found:",struct.formula)
```

Installation

Easy to install. Archive of latest ‘stable’ release.

On Linux:

```
user@computer:> mkdir ~/Bin/python/httk
user@computer:> cd ~/Bin/python/httk
user@computer:> wget http://httk.openmaterialsdb.se/downloads/httk-
    latest.tgz
user@computer:> tar -zxvf httk-latest.tgz
user@computer:> source ~/Bin/python/httk/shell_setup_paths
```

Put the last statement in your .bashrc / .cshrc to always set the paths up correctly.

Installation

Easy to install. Archive of latest ‘stable’ release.

On Mac:

```
user@computer:> mkdir ~/Bin/python/httk
user@computer:> cd ~/Bin/python/httk
user@computer:> wget http://httk.openmaterialsdb.se/downloads/httk-
    latest.tgz
user@computer:> tar -zxvf httk-latest.tgz
user@computer:> source ~/Bin/python/httk/shell_setup_paths
```

Put the last statement in your .bashrc / .cshrc to always set the paths up correctly.

Installation

Easy to install. Archive of latest ‘stable’ release.

On Windows:

- Download and install cygwin

Concluding remarks

- Framework for easy automiatizion of calculations