InChI Certification Suite

Prepared by GGA Software Services LLC, Cambridge, Massachusetts, on November 24, 2010

Overview

The InChI Certification Suite allows independent InChI manufacturers to verify their InChI implementations against the reference data produced by the original InChI/InChIKey implementation. This data is referenced below as the "expected output". The suite can also be used to check the original or third-party InChI implementation for regressions.

System Requirements

Both Windows (XP or higher) and Linux (any modern distributions) are supported in both 32-bit and 64-bit versions. Python 2.X interpreter, preferably version 2.5 or higher, is required in the system. With Python 2.4, users will be able to run the tests, but not to develop them.
Python distributions for Windows are available at http://www.python.org. As for Linux, all modern distributions include Python 2.6 or higher, with one notable exception: RHEL/CentOS 5.5, which has only Python 2.4 available. For developing the test suite properly on RHEL/CentOS 5, the best choice appears to be installing ActivePython precompiled builds (http://www.activestate.com/activepython/downloads).

Deliverables

The deliverables consist of the following two components:

- **inchi-cert**: Certification suite itself
- **inchi-cert-src**: Source data and scripts to generate the certification suite and also to add/remove molecules and molecule classes to/from the suite.

Both parts have the same directory structure. The root directories are:

- `bin` - Python scripts to execute, plus "inchi-1" binary from InChI Trust.
- `lib` - Library binaries: GGA Indigo API (in **inchi-sert-src** only) and InChI API. The source code is not provided, but it can be freely downloaded from: http://inchi-trust.org (for InChI API) and http://ggasoftware.com (for Indigo API).
- `src` - Internal helper scripts.
- `test` - Test set data (Molfile, CML, InChI, InChIKey).

Quick Start: How to Run the Tests from Command Line

The following instructions are suitable for both the **inchi-cert** and **inchi-cert-src** packages. For users who want to only run the tests, rather than alter the test data, **inchi-cert** is enough. Running the tests from the command line requires that some command-line utilities for computation of InChI and InChIKey are available for testing. The InChI utility should be able to take one molecule in Molfile or CML format and produce a file with the InChI code. Similarly, the InChIKey utility should be able to take a file containing a Molfile or an InChI code, and produce a file with InChIKey. By default, the original implementation from the InChI Trust is tested.

1. Go to the "`bin`" folder.
2. Edit the file "`settings.py`", specifying the name and command-line parameters of your command-line application in appropriate places. By default, all settings are tuned for the original `inchi-1` program, which is included in the test suite for convenience.
3. Execute "`python run-the-tests.py`". If your application does not give the expected results, the mismatches will be reported to the standard output.

"`settings.py`" and "`run-the-tests.py`" are explained in greater detail in the next section.

Closer Look at Running the Tests from Command Line

Settings

Along with the internal settings that are not subject to change, the "`settings.py`" file contains definitions of commands to produce InChI from the command line. The definitions have the following structure:

```
inchiTest['windows']['std'] =
  'inchi-1 ##src## ##dst## ##log## nul /NoLabels /AuxNone 1>nul 2>nul'
```

- The value in the first brackets can be "`windows`" and "`linux`". Appropriate values are taken by the "`run-the-tests.py`" script depending on the type of system that the tests run on.
- The values in the second brackets can be: "`std`", which stands for Standard InChI; "`nostereo`", which stands for InChI without the stereo layer; or "fixedh", which stands for InChI without the Mobile H layer.
- The right part is the template for the command to execute. The "`##src##`" and "`##dst##`" are obligatory parts of the template. They define how the names of input and output files are passed to the program. The "`##log##`" part is optional and defines how the name of the log file is passed to the program (if the program supports logging).

Similarly, the definitions for InChIKey program have the following structure:

```
inchiKeyTest['windows']['molfile'] =
  'inchi-1 ##src## ##dst## ##log## nul /NoLabels /AuxNone /Key 1>nul 2>nul'
```

The value in the second brackets in this case can be "`molfile`" or "`inchi`". The first option means that the InChiKey-s are produced from Molfiles, while the second option means that they are produced directly from InChI codes. The default setting for the "`molfile`" mode is the "inchi-1" program, and the default setting for the "`inchi`" mode is a small Python command-line wrapper around the InChI API. The wrapper was created because the "inchi-1" program itself does not provide an option to generate an InChiKey directly from InChI.
Note: While running the tests is possible under Python 2.4, the InChI API wrapper requires `ctypes` module, and will only work under Python 2.5 or higher. To avoid runtime errors when running the suite against the original inchi-1 program, users may disable the tests for InChI->InChIKey conversion, providing "`--inchikey-types=molfile`" flag to "`run-all-tests.py`" (see the detailed options below).
Note: Running the tests against the original "inchi-1" program on Windows works considerably slower than running them on Linux; in fact, it is at least 5 times slower. However, the slowdown is specific to the original implementation. The performance of Windows tests of an alternative InChI implementation will be good as long as the alternative implementation is fast on Windows.

Options

"`run-the-tests.py`" script accepts the following command-line options:

- `--molecule-classes=class1,class2,...,classN`
  Comma-separated list of molecule classes to test. If not specified, the tests will be performed for all available classes.
- `--inchi-input-formats=molfile`
  `--inchi-input-formats=cml`
  `--inchi-input-formats=molfile,cml`
  This option is for choosing whether to test the InChI generation from Molfiles, or CML files, or both. If the key is omitted, both formats are tested.

- `--inchikey-types=inchi`
  `--inchikey-types=molfile`
  `--inchikey-types=inchi,molfile` *(the default)*
  `--inchikey-types=''`
  This options is for choosing whether to test the InChIKey-s produced from InChI, or the InChIKeys produced from Molfiles, or both (this is the default). Empty value means that no InChIKey tests are performed, which is relevant for somebody who wants to run only the tests for InChI.
- `--stdout`
  Write the full information about each encountered mismatch to the standard output. If it is not specified, only a short notice with a link to a "`.cmp`" file is printed.
- `--list`
  Write the list of molecule classes and then the lists of InChI and InChIKey types, and then terminate. Perform the tests for both Molfile and CML input (the default is Molfile only).
- `--time`
  Write time measurement data for each test to standard output. The data includes the minimal time of generating InChI for a single molecule, the maximal time, the mean value, and the standard deviation.
- `--output=`*`<output file>`*
  Duplicate the program output to a file.
- `--errorfile=`*`<error report file>`*
  Write a separate report that only contains information about failed tests.

Results

Once the "`run-the-tests.py`" has finished running, a user can examine its output to know the names of the structures that caused the mismatches (if any). The full information is provided in the following files:

`test/inchi/`*`<class>`*`/tests/molfile/`*`<class>`*`-`*`<std|nostereo|fixedh>`*`-all.cmp`
Information about all Molfiles whose calculated InChI codes do not match the expected output. "class" is the name of the molecule class; "std" or "nostereo" or "fixedh" is the type of the InChI code.

`test/inchi/`*`<class>`*`/tests/molfile/`*`<class>`*`-`*`<std|nostereo|fixedh>`*`-`*`xxxx`*`-`*`yy`*`.cmp`
Information about each particular Molfile whose InChI code does not match the expected output. *xxxx* is the number of the molecule within the class, and *yy* is the number of the permutation.

`test/inchi/`*`<class>`*`/tests/molfile/`*`<class>`*`-`*`xxxx`*`-`*`yy`*`.mol`
The Molfile containing the molecule that caused the problem described in the file above.

`test/inchi/`*`<class>`*`/tests/cml/`*`<class>`*`-`*`<std|nostereo|fixedh>`*`-all.cmp`
Information about all CML files whose calculated InChI codes do not match the expected output. "class" is the name of the molecule class; "std" or "nostereo" or "fixedh" is the type of the InChI code.

`test/inchi/`*`<class>`*`/tests/cml/`*`<class>`*`-`*`<std|nostereo|fixedh>`*`-`*`xxxx`*`-`*`yy`*`.cmp`
Information about each particular CML file whose InChI code does not match the expected output. *xxxx* is the number of the molecule within the class, and *yy* is the number of the permutation.

`test/inchikey/`*`<class>`*`/tests/molfile/`*`<class>`*`-molfile-all.cmp`
Information about all Molfiles whose Standard InChIKey codes do not match the expected output.

`test/inchikey/`*`<class>`*`/tests/molfile/`*`<class>`*`-molfile-`*`xxxx`*`-`*`yy`*`.cmp`
Information about each particular Molfile whose Standard InChIKey code does not match the expected output. *xxxx* is the number of the molecule within the class, and *yy* is the number of the permutation.

`test/inchikey/`*`<class>`*`/tests/molfile/`*`<class>`*`-`*`xxxx`*`-`*`yy`*`.mol`
The Molfile containing the molecule that caused the problem described in the file above.

`test/inchikey/<class>/tests/inchi/<class>-inchi-all.cmp`
Information about all Standard InChI codes whose InChIKey codes do not match the expected output.

`test/inchikey/<class>/tests/inchi/<class>-inchi-xxxx.cmp`
Information about each particular Standard InChI code whose InChIKey does not match the expected output. *xxxx* is the number of the molecule within the class.

`test/inchikey/<class>/tests/inchi/<class>-xxxx.inchi`
The InChI that caused the problem described in the file above.

Molecule Classes Present in the Test Set

Classes Focused on Particular Structure Features

1. **basic**: Some basic molecules of no particular interest
2. **ali_ring_het_{5,6,7,8}**: Aliphatic rings of sizes 5,6,7,8 with heteroatoms
3. **arom_het_{5,6,7,8}**: Aromatic rings of sizes 5,6,7,8 with heteroatoms
4. **arom_fused**: Fused aromatic rings
5. **elements**: Elements of the periodic table
6. **element_collections**: Collections of all elements from the periodic table
7. **big_charges**: Structures with big charges (either positive or negative)
8. **tetrahedral**: Chiral centers
9. **stereo_either**: “Either” stereo bonds (only Molfiles)
10. **cistrans**: Cis-trans double bonds
11. **cumulenes**: Allenes and cumulenes
12. **isotopes_h**: Deuterium and tritium
13. **isotopes_nonh**: Other isotopes
14. **radicals**: Atom radicals
15. **valence**: Marked and/or nonstandard valence
16. **resonance_{chn,rng}**: Chemical resonance (chain-specific and ring-specific)
17. **dipoles:** Dipoles
18. **salts**: Salts
19. **tau_{simple,charge,ring}**: Tautomers: simple, ring-specific, and chain-specific
20. **explicit_h**: Explicit hydrogens
21. **spatial**: Spatial (3D) structures

Note: While most classes are represented in both Molfile and CML formats, the **stereo_either** class is represented in Molfile format only. The CML representation of the structures in this class is impossible due to limitations of the CML format.

Classes Containing Complex Structures

1. **large**: Various large structures
2. **symmetric**: Various structures with high symmetry
3. **organometallics**: Organometallic structures
4. **disconnected**: Various disconnected structures
5. **disconnected_stereo**: Various disconnected structures with stereochemistry
6. **tricky_stereo**: Structures with legal but unsupported stereo configurations
7. **complex_stereo**: Structures with complex stereochemistry

Other Classes

1. **known_bugs**: Previously known molecules on which the InChI program fails
2. **invalid_stereo**: Molecules with invalid stereo configurations

Certification Package Structure

Test Folder

test/inchi/<*class*> folders

- `input/cml/<class>-all.cml`: input structures in CML format
- `input/molfile/<class>-all.sdf`: input structures in Molfile format
- `output/std/<class>-std-all.inchi`: expected standard InChI output
- `output/nostereo/<class>`-nostereo-all.inchi: expected "nostereo" InChI output
- `output/fixedh/<class>-fixedh-all.inchi`: expected "fixedh" InChI output
- `logs:` Logs (if any) of a command-line program being tested are placed into this folder.
- `test:` Mismatches (if any) of the results obtained by the program with the expected results are placed into this folder. The structure of this folder is explained above in the "Results" section.

test/inchikey/<class> folders

- `input/inchi/<class>-std-all.inchi`: input InChI codes
- `input/molfile/<class>-all.sdf`: input structures in Molfile format
- `output/std/<class>-std-all.inchikey`: expected standard InChIKey output
- `logs`: Logs (if any) of a command-line program being tested are placed into this folder
- `test:` Mismatches (if any) of the results obtained by the program with the expected results are placed into this folder. The structure of this folder is explained above in the "Results" section.

test/settings folder

The "settings" folder contains the precalculated data for the molecule classes, stored in files called "<*molecule-class*>.cfg". For each class, the amount of structures is stored along with the list of permutations for each structure in the class.

Bin folder

The "bin" folder contains Python scripts needed to run the tests. In the certification package, these are just "run-the-tests.py", "settings.py", and "inchikey.py", while in the source package the "bin" folder also contains the scripts for altering the test data.

Lib folder

The "lib" folder contains only library binaries for Windows and Linux operating systems, both 32-bit and 64-bit. Two libraries are present there:

- `libinchi`, whose functionality of InChIKey generation is tested in the suite.
- `libindigo` (included only in the source package), used to compute permutations of structures. It is used twice: first by the "`add_molecule.py`" script to generate unique permutations, and second by the "install" script to restore the permuted molecules using the saved permutation lists.

Src Folder

The “src” folder contains helper Python scripts that are used internally by the scripts in the “bin” folder.

Managing the Test Set

This section applies exclusively to the **inchi-cert-src** package.

Scripts

The scripts are located in the “`bin`” folder.

- `add_moleculeclass.py --name=`*`<molecule class>`*
  Adds a new molecule class with a given name.
- `rename_moleculeclass.py --old-name=`*`<old name>`*` --new-name=`*`<new name>`*
  Renames the specified molecule class.
- `delete_moleculeclass.py --name=`*`<molecule class>`*
  Deletes the specified molecule class.
- `add_molecule.py --path=`*`<path to molfile> OR`*` --smiles=`*`<SMILES string>`*` --molecule-classes=`*`<molecule classes>`*
  Adds the given molecule to the test set. The molecule will be added to each of the specified molecule classes. The molecule will be automatically assigned a unique 4-digit number.
  The random (but different) permutations (100 maximum) of the input molecule are automatically generated at this stage. The CML files, InChI files, and InChIKey files are also produced.
  Note: Although the permutations are calculated, the permuted Molfiles and CML files are not saved. The permutations are encoded to a “.cfg” file, which is used at the installation stage to generate the actual Molfiles and CML files.
  Note: Molecule must be specified with a `--smiles` key or with a `--path` key, but not both. If the input is a SMILES string, it will be automatically converted to a Molfile. The SMILES string itself will not be included in the test suite.
- `add_sdf.py --path=`*`</path/to/file.sdf> --molecule-classes=<molecule classes>`*
  Adds all structures from the given SDF file to the given molecule class (or classes).
- `delete_molecule.py --molecule-class=`*`<molecule class>`*` --number=`*`<4-digit number>`*
  Deletes the specified molecule from the specified class.

-
  ```
  generate_inchi.py [--molecule-classes=<molecule classes>]
                    [--inchi-types=<std,nostereo,fixedh>]
  ```
  Generates Standard or “nostereo” or “fixedh” InChI codes (or all of them) for the specified molecule classes. The list of molecule classes is comma-separated. If no `--molecule-classes` is specified, all present classes will be processed. If no `--inchi-types` is specified, all InChI types will be calculated.
  Note: The cases when a user really needs to run `generate_inchi.py` are rare. Normally, all of the calculations are done automatically by other scripts. Running generate_inchi.py manually is needed only (i) to generate reference InChI codes for some new “InChI type” added in the settings.py file, or (ii) to regenerate reference InChI codes if the original “inchi-1” utility has been changed to a new version.

-
  ```
  generate_inchikey.py  [--molecule-classes=<molecule classes>]
                        [--inchikey-types=<molfile,inchi>]
  ```
  Generates InChIKey codes for selected molecule classes from Molfile or InChI input (or both). The list of molecule classes is comma-separated. If no `--molecule-classes` is specified, all classes will be processed. If no `--inchikey-types` is specified, the InChIKeys will be generated from both Molfiles and InChI codes.

Note: The note given above for generate_inchi.py script applies to this script too.

- `extract-sdf.py [--molecule-classes=`*`<molecule classes>`*`] [--number=`*`<4-digit number>`*`] [--path=`*`<destination path>`*`]`
  Extracts molecules from specified classes to Molfiles. If a "`--molecule-class`" and a "`--number`" are given, a single molecule from the specified class gets extracted to a molfile. If no "`--number`" is given, all of the molecules from the class (i.e. the whole SDF file) are extracted. If neither "`--molecule-class`" nor "`--number`" are given, all classes are extracted.
- `install.py --path=`*`<path to an empty or non-existent folder>`*
  Deploys the test set to the specified folder. That is, a directory containing the **inchi-cert** distribution is produced.