IUPAC International Chemical Identifier (InChI)

InChI version 1, Software version 1.06 pre-release PR3

Release Notes

Last revision date: September 05, 2020

Summary

This document introduces the pre-release of InChI Software version 1.06 PR3 of 2020.

The current version of InChI Identifier is 1; the current stable version of the InChI software is 1.05 which is planned to be replaced with v. 1.06.

InChI Software v. 1.06 includes several additions and modifications to previous versions; it is a combination of “bugfix release” and “feature release”.

What is new:

- security-related bugfixes;
- a number of other bugfixes and minor improvements;
- experimental support for pseudo element (Zz, or "star") atoms;
- modified experimental support of InChI/InChIKey for regular single-strand polymers (explicit pseudo atoms are used);
- minor update to InChI API Library;
- optional support of Intel(R) Threading Building Blocks scalable memory allocators;

- several convenience features (for example, ability to use less strict rules for assigning tetrahedral stereo in large rings) and corresponding software options.

A brief description of novel and changed features is given below. For more details, please check the documents included in this distribution (all in PDF format):

- CHANGELOG (list of changes relative to the previous release)
- KNOWNISSUES (list of known issues)
- LICENCE (IUPAC/InChI Trust software licence)
- InChI_UserGuide (software user's guide)
- InChI_API_Reference (programmer's reference for API library)
- InChI_TechMan (technical manual on InChI identifier)

Security-related fixes

A number of fixes have been made to eliminate issues which may occasionally lead to security compromise.

Since v. 1.05 release, there appeared several requests concerning possible problems. A typical scenario is that invalid input (namely, an input which looks like a normal Molfile or InChI string, but actually contains invalid fragments or garbage of any origin) in some circumstances may cause InChI software memory corruption and crash, instead of proper behavior of issuing diagnostic message and rejecting such an input. Theoretically, such crashes may compromise security of services which internally use InChI software (if these services, for any reason, do not check themselves their inputs for possible invalidity/malicious nature). As InChI Software got wide popularity and became a backend engine used internally by many production environments and services, these issues could not be neglected.

In particular, fixed were:

- 20 issues reported by Google AutoFuzz project;
- 103 issues reported by Google's Ian Wetherbee;
- several similar issues reported by Cure53 company;
- number of several similar or less similar issues from own backlog or user reports (in particular, reported by Andrew Dalke and by Burt Leland issue of improper handling H isotope with abnormally high mass difference).

Other fixes

Several issues that related to core InChI functionality have been fixed in v. 1.06 pre-release.

In particular, there were:

- Fixed 2 "no-calc" issues.
  These are the cases where InChI Software v. 1.05 could not produce InChI string for given input Molfile.
- Fixed 1753 renumbering issues.
  These are the cases where InChI strings produced by InChI Software v. 1.05 change upon renumbering atoms of input Molfiles. Mostly they are related to the bug in normalization for some structures containing acidic hydroxy group at cationic heteroatom center.

Some similar issues still remain active. For more details see the section "Known issues" below.

Several minor bugs are fixed, for example:

- bug of not recognizing olean-like spiro chirality;
- bug in polymer CRU canonicalization at comparing seniority of 'junior members' for various possible options of caps attachment;
- bug for polymers which may appear in multipliers in brutto formula;
- several bugs in AuxInfo output (some general and some specific to IXA API);
- critical race condition bugs pointed out by Karl Nedwed and by John Salmon;
- bug which caused crash on reading some V3000-formatted large molecules.

For more information, see CHANGELOG document accompanying this distribution.

New and updated features

Support of pseudo element (Zz) atoms

Zz pseudoelement atom (“star atoms”) is a generic placeholder designating undefined/unknown/variable nature entity.

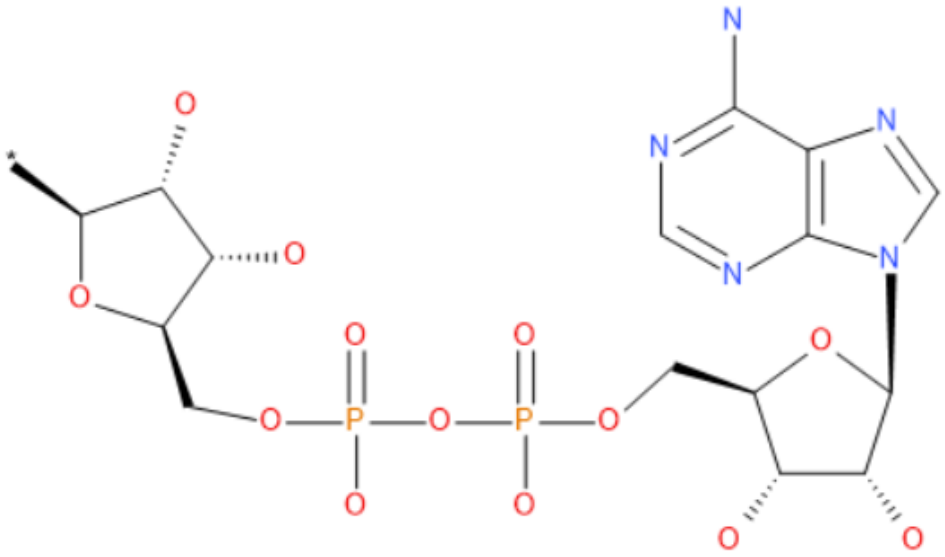
Exact meaning of Zz atoms, as well as the rules of treating them, is up to application chemist/programmer. One specific exception is a case of pair of Zz atoms surrounding polymeric CRU (constitutional repeat unit), that is, sitting at the ends of bonds crossing CRU brackets. These Zz atoms are considered as undefined-nature end groups surrounding polymer repeat units and are treated according to rules common in polymer chemistry, see below.

Zz is considered an univalent pseudo element (so Zz atoms are always terminal) having the least possible InChI seniority (that is, Zz atoms will always have the maximal available canonical numbers). The stereo at the atoms connected to Zz is disabled by default (may be enabled by option *SAtZz*).

On input, pseudo element may be represented by either symbol “Zz” or symbol ‘*’ ; output always contains Zz as * is reserved in InChI for other purposes

By default, usage of Zz atoms is allowed for handling polymers (which requires specifying *Polymers* switch of inchi-1/API).

To allow usage of Zz atoms out of polymer context, option *NPZZ* should be specified. Shown below is the example of using non-polymeric Zz for adenosinediphosphoribosyl group, structure CHEBI:22259




```
InChI=1B/C15H22N5O13P2Zz/c16-12-7-13(18-3-17-12)20(4-19-7)14-10(23)8(21)5(31-
14)1-29-34(25,26)33-35(27,28)30-2-6-9(22)11(24)15(36)32-6/h3-6,8-11,14-15,21-
24H,1-2H2,(H,25,26)(H,27,28)(H2,16,17,18)/t5-,6-,8-,9-,10-,11-,14-/m1/s1
InChIKey=HGZGKBRIZICMCZ-UQZBLMPMBA-N
```

For the details of API and inchi-1 options related to handling pseudo atoms, please consult UserGuide and InChI_API_Reference documents included in this distribution.

Please note that caution is necessary while using Zz atoms due to inherently indefinite nature of pseudo element. The consistency of Zz interpretation is by no means guaranteed, as it is by design mostly left to user/application programmer, and various side effects may appear. To emphasize this, appearance of Zz atoms, both in non-polymer and polymer context, is considered experimental feature and corresponding InChI receives prefix "InChI=1B".

Modifications in support of regular single-strand polymers

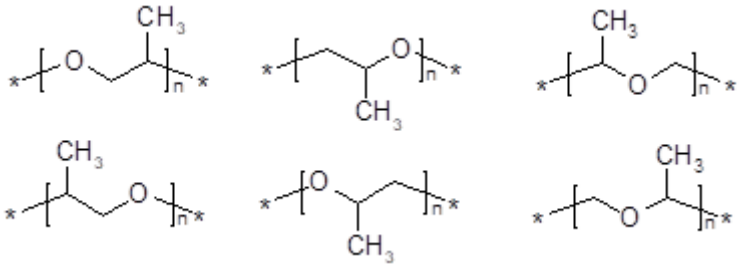
Polymer treatment in v. 1.06 is updated to extend functionality and accommodate Zz atoms, as well as to fix bugs and to account for users feedback.

Zz (“star”) atoms at the ends of bracket-crossing bonds are now considered, as is common to polymer chemistry, as the end groups of undefined nature. Consequently, their presence makes InChI algorithm to perform so-called CRU (Constitutional Repeat Unit; aka SRU, Structural Repeat Unit) “frame shift analysis” to ensure canonicalization of repeat unit(s) in infinite chain. Due to frame shift, bracket-crossing bonds may be reattached to CRU atoms resulting in changed bonding scheme.

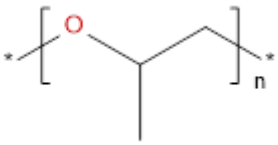
Actually, Zz atoms were partially introduced in v. 1.05 for handling polymers, but their treatment has been implemented internally and expressed in “latent” mode: possible reattachment and presence of Zz atoms were “hidden” in InChI string and become evident only on back-conversion of InChI to structure.

As in InChI v. 1.06 “indefinite-nature” caps of CRU is represented explicitly by Zz atoms, frame shift (if any) occurs upon generation of InChI from structure, so that bonding scheme of input molecule is corrected to adjust to canonical CRU form.

For example, all the structures



are converted to the same canonical structure with explicit Zz atoms (appeared as “Zz” in InChI string but shown as common to chemists asterisk on the scheme):



which results in the same identifier for all inputs, namely:

```
InChI=1B/C3H6OZz2/c1-3(2-5)4-6/h3H,2H2,1H3/z101-1-4(6-4,5-2)
InChIKey=HWXRSRVZRSEKDJ-IDWFELBQBA-N
```

The changed approach means that InChI strings for all polymers, containing CRU with indefinite-nature caps, do change.
Thus, for the above example, v. 1.05 identifiers were:

```
InChI=1B/C3H6O/c1-3-2-4-3/h3H,2H2,1H3/z101-1-4(2,3,2,4,3,4)
InChIKey=GOOHAUXETOMSMM-KUWDYTNTBA-N
```

To retain compatibility with previous version, the new option *Polymers105* added. It instructs the API/inchi-1 to treat polymer data as in v. 1.05, that is, hide explicit Zz atoms. It is planned that this option will be eliminated in future, leaving explicit-pseudo atoms approach the sole mode.

Some limitations of polymer analysis were eliminated.

Canonicalization of CRU containing inner repeats may now include removal of repeats by folding to the "least repeating unit".
A simple example:
`*-[-CH2CH2-]n-*`
is folded to
`*-[-CH2-]n-*`.
This is not the default mode of action (as many chemists would not expect that polyethylene is converted to polymethylene); folding is activated by specifying new software option *FoldCRU* which works for both inchi-1 and API (another new option in v. 1.06 is *NoFrameShift* which turns off frame shift) .

Copolymers represented as combination of source-based and structure-based subunits are now allowed.

InChI API

InChI Software v. 1.06 includes minor update of INCH API.

Added are the following procedures :

```
IXA_INCHIBUILDER_SetOption_Timeout_Milliseconds()
IXA_MOL_GetBondOtherAtom()
IXA_MOL_ReserveSpace()
IXA_MOL_CreatePolymerUnit ()
IXA_MOL_SetPolymerUnit ()
IXA_MOL_GetPolymerUnitId()
IXA_MOL_GetPolymerUnitIndex()
```

The behavior of several API procedures is slightly changed to improve their usability and account for newly added features.

Also improved are some inner details of API, in particular, performance of IXA memory allocations is enhanced.

Note that InChI "modularized version of old" API (the set of functions using `INCHIGEN` object) is now frozen. These procedures still work and are retained for compatibility reasons, but will not receive further development.

For the details of current InChI API, please consult InChI_API_Reference document included in this distribution.

Intel TBB scalable memory allocator support

InChI Software v. 1.06 provides optional support of Intel(R) Threading Building Blocks (TBB) scalable memory allocators (SMA) which may improve performance of InChI library in multi-threading program environment.

Outline

Upon using InChI library in multi-threading environment, memory allocation can become a serious performance bottleneck due to locking issues. As threads may compete for a global lock related to a single global heap, program's behavior is not scalable and speed may even degrade if number of processor cores increases. Scalable memory allocators resolve the issues by relying on more sophisticated data structures/logics.

Intel TBB is free software package available for both Windows and Linux, licensed under the Apache License, Version 2.0, see file LICENSE in TBB sub-directory of distribution (there is also a commercial counter-part; consult https://www.threadingbuildingblocks.org).

TBB offers a "proxy" method to automatically replace C functions for dynamic memory allocation with its own scalable memory allocators to largely avoid contention (there exist other, non-automatic ways of using TBB allocators, which we do not touch).

This method may optionally be used with InChI Software library and, in some cases, may provide performance gain (but your mileage may vary, so try and check).

For additional info on Intel scalable memory allocator and Intel TBB in general, please visit https://www.threadingbuildingblocks.org/documentation
or consult print literature, e.g., open access book "Pro TBB. C++ Parallel Programming with Threading Building Blocks" by Michael Voss, Rafael Asenjo, and James Reinders: https://link.springer.com/book/10.1007/978-1-4842-4398-5

How to use

Use SMA-aware versions of libinchi dynamic libraries (`libinchi.dll` under Windows and `libinchi.so` under Linux), either supplied with this distribution or compiled from sources.

To run the application calling libinchi, ensure that it has access to Intel TBB SMA dynamic library.

Under Linux, the easiest way is to load SMA libraries at application load time using the `LD_PRELOAD` environment variable (another option is by linking the main executable file with the proxy library). As loader must be able to find the proxy library and the allocator library, one may include the directory with libraries in the `LD_LIBRARY_PATH` environment variable or add it to `/etc/ld.so.conf`.

Example:

```
export
LD_PRELOAD="/opt/intel/compilers_and_libraries_2020.1.217/linux/tbb/lib/intel64_
lin/gcc4.8/libtbbmalloc_proxy.so.2
/opt/intel/compilers_and_libraries_2020.1.217/linux/tbb/lib/intel64_lin/gcc4.8/l
ibtbbmalloc.so.2"

./mol2inchi chembl_26.sdf -Threads:4 -Perthread:1024 -NoWarnings -AuxNone -Key
>/dev/null 2>m2i-4-TBB.log
```

Under Windows, the simplest way is to copy `tbbmalloc_proxy.dll` and `tbbmalloc.dll` to the directory containing program and libinchi.dll. Alternatively, you may put directory containing SMA library(-ies) to PATH environment variable. Note that the libraries for 32-bit and 64-bit Windows are different, so be careful to properly match “bitness” of involved binaries.

Precompiled SMA-aware InChI library binaries are supplied in this distribution.

Intel TBB SMA library binaries are also supplied, for convenience, see packaging notes below (they may also be downloaded from Intel TBB site).

As an alternative to using precompiled InChI library SMA-enabled binaries, you may build them. For that purpose, uncomment the line

```
#define USE_TBB_MALLOC 1
```

in the file `mode.h` and build `ibinchi.dll` under Windows as usual, or build `libinchi.so` under Linux using supplied makefile `makefile_with_tbb`.

Other

Structure perception option *LooseTSACheck*

A new InChI structure-perception (i.e., available for generation of Standard InChI) option *LooseTSACheck* relaxes the strictness of tetrahedral stereo ambiguity check, for in-ring atoms, thus getting stereo descriptor instead of undefined mark ‘?’.

It accounts for now common case of large molecules/large cycles, drawn and then automatically 'cleaned' by chemoinformatics software, see the example below.

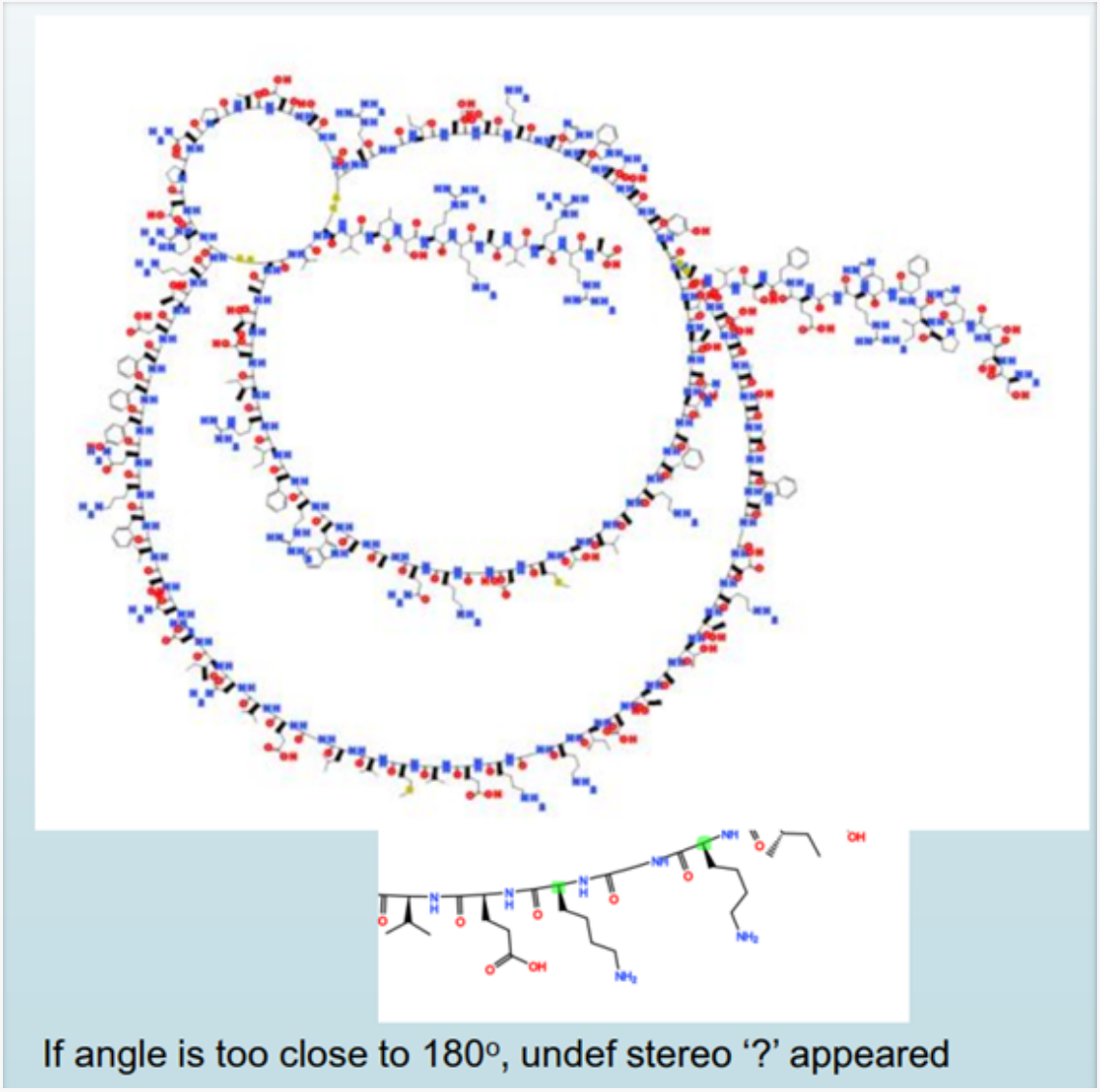


Figure 1. Auto-cleaned by drawing software large-ring molecule may get undefined stereo '?' mark(s) due to too strict InChI's tetrahedral stereo ambiguity check. Option LooseTSACheck fixes this issue and produces definite stereo descriptor.

The option is available for inchi-1 executable and InChI API, see "User's Guide" and "InChI API Reference", resp.

New timeout switch *WMnumber*

New convenience switch *WMnumber* is added (already available one *Wnumber* sets value in seconds).

It sets InChI calculation timeout in milliseconds (strictly requires long int number), thus adding finer granularity which is necessary for long runs on multi-million-input datasets.

Changes to inchi-1 executable

Several convenience changes were made to inchi-1 executable.

For the details, please consult "UserGuide" document in this distribution.

Testing

Regression testing

As there were several minor fixes/changes made after software release v. 1.05, to ensure compatibility with previous release the new software has been extensively regression-tested against standard and non-standard InChIs generated with v. 1.05 Software, in both Windows and Linux environments, for the various datasets, with the various option combinations.

The difference in the results has been found extremely small, as is illustrated by the table below.

The datasets here and below are ChEMBL v. 26 (1,940,733 molecules) and PubChem Compound (102,710,107 molecules), both downloaded from the respective sites on May 2020.

Dataset	**Different InChI's**	**Different, %**
Standard InChI		
ChEMBL	0	0
PubChem Compound	4555	0.004
Non-standard InChI (FixedH RecMet)		
ChEMBL	0	0
PubChem Compound	4555	0.004
Non-standard InChI (SLUUD)		
ChEMBL	27	0.001

The differences are due to introduced bug fixes. For example, the difference in 27 InChI strings out of 1,940,733 for ChEMBL v. 26 is entirely due to v. 1.06 fix for bug of not recognizing olean-like spiro chirality (seen if *SLUUD* option is active); the major part of difference in InChI strings for PubChem, 4456 entries, is due to v. 1.06 fix for "renumbering instability" bug.

InChI round-trip test

The success rate for conversion of InChI to structures was measured using Rtrip option of test_ixa program or two-pass (structure to InChI followed by InChI to structure) execution of inchi-1. In this round trip, InChIs generated from Molfile-format records were then used to restore structures and re-create InChIs from those structures, via corresponding API calls; original and final InChIs were then compared.

The representative results are shown in the table below.
The figures are slightly improved in comparison to those for v. 1.05.

Dataset	Success rate, %
Standard InChI	
ChEMBL	99.98
PubChem Compound	99.95
Non-standard InChI (FixedH RecMet)	
ChEMBL	99.98
PubChem Compound	99.94

Known issues

The currently known issues have been found through internal tests including ~100M compounds of PubChem Compound and some other datasets, via multiple runs with different software options and extensive atomic renumbering of input data.

The current issues include:

- 5 cases of "no-calc" issue (inability of Software to generate InChI string);
- 547 cases of "renumbering instability" (change of InChI string after renumbering of atoms in molecule).

The previous v. 1.05 demonstrated higher number of issues. This v. 1.06 pre-release fixes:

- 2 "no calc" cases;
- 1753 "renumbering instability" cases.

All the past and present issues are compiled, with corresponding PubChem CID numbers, in KNOWNISSUES document accompanying this distribution.

Distribution package

Binaries

This package includes 'command line' InChI executable and InChI API Library binaries (32 and 64 bit versions are supplied for both Windows and Linux).

Also included is winchi-1.exe, a graphical Windows application (a 32 bit version which will also run under 64 bit Windows).

The binaries are placed according the following tree.

INCHI-1-BIN

windows

32bit

winchi-1.exe	InChI graphical Windows application
inchi-1.exe	InChI stand-alone command line executable, 32 bit

dll

libinchi.dll	InChI dynamic-link library, 32 bit

with_tbb

libinchi.dll InChI dynamic-link library, 32 bit, with Intel TBB memory allocators support

64bit

inchi-1.exe InChI stand-alone command line executable, 64 bit

dll

libinchi.dll InChI dynamic-link library, 64 bit

with_tbb

libinchi.dll InChI dynamic-link library, 64 bit, with Intel TBB memory allocators support

linux

32bit

inchi-1.gz InChI stand-alone command line executable, 32 bit; gzipped

so

libinchi.so.1.06.00.gz shared library for InChI API, 32 bit; gzipped

64bit

inchi-1.gz InChI stand-alone command line executable, 64 bit; gzipped

so

libinchi.so.1.06.00.gz shared library for InChI API, 64 bit; gzipped

Note that InChI stand-alone executable inchi-1 does not require dll/so libraries.

Source codes and demo programs

InChI Software binaries are placed in the file/directory INCHI-1-BIN.

Example data files are placed in the file/directory INCHI-1-TEST.

Documentation is placed in the file/directory INCHI-1-DOC.

InChI Software source codes are placed in the file/directory INCHI-1-SRC. This file/directory also contains examples of InChI API usage, for C ('inchi_main', 'mol2inchi', 'test_ixa', see projects for MS Visual Studio 2015 in 'vc14' and for gcc:Linux in 'gcc' subdirs) and Python 3 ('python_sample'). Also supplied are InChI API Library source codes and related projects/makefiles.

The projects/makefiles necessary to build inchi-1 executable and demo programs are located within corresponding directories, as well as the source codes specific for these apps (see below). Note that a part of code which forms a common codebase is placed in the directory INCHI-1-SRC/INCHI_BASE/src.

To ensure proper build of InChI applications/library, the directory structure below (a tree under INCHI-1-SRC) should be preserved. Note also that though InChI library 'libinchi' may be build using its own projects/makefiles (under INCHI-1-SRC/INCHI_API/libinchi/), it is automatically co-created upon building of any demo program.

The details of source tree is as follow.

INCHI-1-SRC/INCHI_BASE

src

C source files - common codebase used to build both InChI Library and inchi-1 executable

INCHI-1-SRC/INCHI_EXE

bin

directory where the binaries of inchi-1 executable are created/stored

inchi-1

a home directory for inchi-1 command-line executable

src

C source files specific for inchi-1 executable

gcc

gcc makefiles for inchi-1 executable (Linux, 64- and 32-bit)

vc14

MS VS2015 project for inchi-1 executable (Windows)

INCHI-1-SRC/INCHI_API

bin

directory where the binaries (of both library and all demo programs) are created/stored

bin2

alternative directory where the binaries (of both library and all demo programs) are created/stored

libinchi

a home for InChI Software Library

src

C source files specific for InChI Software Library

gcc

gcc makefiles for libinchi library (Linux so)

vc14

MS VS2015 project for libinchi library (Windows dll)

demos

a home for demo programs calling InChI library (API)

inchi_main

inchi_main demo program

src

C source files specific for inchi_main demo

gcc

gcc makefiles for inchi_main demo program (Linux)

vc14

MS VS2015 project for inchi_main demo program (Windows)

mol2inchi

mol2inchi demo program

src

C source files specific for mol2inchi demo program

gcc

gcc makefiles for mol2inchi demo program (Linux)

vc14

MS VS2015 project for mol2inchi demo program (Windows)

test_ixa

test_ixa demo program

src

C source files specific for test_ixa demo program

gcc

gcc makefiles for test_ixa demo program (Linux)

vc14

MS VS2015 project for test_ixa demo program (Windows)

python_sample

Python 3 source files specific for Python demo program

Please refer also to 'readme.txt' files in the directories.

Other

The documentation (Release Notes; InChI Technical Manual; InChI User Guide; InChI API Reference) in PDF format is supplied in the INCHI-1-DOC section of this distribution package.

Test data are supplied in the INCHI-1-TEST file/directory.

Acknowledgements

We are grateful to many people and organizations who are continuously supporting InChI development, in various forms.

Thanks are due to those who provided feedback, reported on problems and participated in discussions, namely,
Gerd Blanke, Evan Bolton, Yulia Borodina, Steve Boyer, Cure53 team, Andrew Dalke, Igor Filippov, Google AutoFuzz team and Ian Wetherbee, Wolf-Dietrich Ihlenfeldt, Burt Leland, Peter Linstrom, Daniel Lowe, John Mayfield, Karl Nedwed, Noel O'Boyle, Dmitry Redkin, John Salmon, Roger Sayle, Paul Thiessen, Lutz Weber, Egon Willighagen, Andrey Yerin, and to many others who specifically contributed to the current release/testing, especially Dmitrii Tchekhovskoi.