Standard InChI/InChIKey API Reference

This is the release of the IUPAC standard International Chemical Identifier with InChIKey, version 1, software version 1.02.
(http://www.iupac.org/projects/2000/2000-025-1-800.html
http://www.iupac.org/inchi).

The release conforms to standard InChI and standard InChIKey definitions (see Standard InChI User's Guide) as established by IUPAC InChI Subcommittee at its September 15, 2008 meeting (for details, see http://sourceforge.net/mailarchive/message.php?msg_name=20081002155703.FXZB29597.aamtaout03-winn.ispmail.ntl.com%40ALAN).

Below is a brief description of standard InChI/InChIKey API functions (for the related data structures/parameters and more details see inchi_api.h header file in the standard InChI software source code).

Standard InChI generation API - modularized

Seven new API functions were added to InChI software version 1.01 interface to InChI library. These functions are described in inchi_api.h header file.

The main purpose was to modularize the process of InChI generation by separating normalization, canonicalization, and serialization stages. Using these API functions allows, in particular, checking intermediate normalization results before performing further steps and getting diagnostics messages from each stage independently.

INCHIGEN_HANDLE STDINCHIGEN_Create(void)
Standard InChI Generator: create generator
Returns: a handle of the InChI generator object or NULL on failure
Note: the handle serves to access the internal object, whose structure is invisible to the user (unless the user chooses to browse the InChI library source code which is open).

int STDINCHIGEN_Setup(INCHIGEN_HANDLE HGen, INCHIGEN_DATA *pGenData, inchi_Input * pInp)
Standard InChI Generator: initialization stage (storing a specific structure in the generator object)
Note: **INCHIGEN_DATA** object contains intermediate data visible to the user, in particular, the string accumulating diagnostic messages from all the steps.

int STDINCHIGEN_DoNormalization(INCHIGEN_HANDLE HGen, INCHIGEN_DATA * pGenData)
Standard InChI Generator: perform structure normalization
Note: **INCHIGEN_DATA** object explicitly exposes the intermediate normalization data, see below.

int STDINCHIGEN_DoCanonicalization(INCHIGEN_HANDLE HGen, INCHIGEN_DATA * pGenData)
Standard InChI Generator: perform structure canonicalization

int STDINCHIGEN_DoSerialization(INCHIGEN_HANDLE HGen, INCHIGEN_DATA * pGenData, inchi_Output * pResults)
Standard InChI Generator: perform InChI serialization

void STDINCHIGEN_Reset(INCHIGEN_HANDLE HGen, INCHIGEN_DATA * pGenData, inchi_Output * pResults)
Standard InChI Generator: reset (use before calling **STDINCHIGEN_Setup(…)** to start processing the next structure and before calling **STDINCHIGEN_Destroy(...)**)

void STDINCHIGEN_Destroy(INCHIGEN_HANDLE HGen)
Standard InChI Generator: destroy the generator object HGen. Important: make sure **STDINCHIGEN_Reset(…)** is called before calling **STDINCHIGEN_Destroy(…)**.

The functions use exactly the same **inchi_Input** and **inchi_Output** data structures as other InChI API functions do. However, a new data structure, **INCHIGEN_DATA**, has been added to expose the normalization results (see inchi_api.h header file).

A typical process of InChI generation with this API calls is as follows.

1. Get handle of a new InChI generator object:
   HGen = STDINCHIGEN_Create();
2. read a molecular structure and use it to initialize the generator:
   result = STDINCHIGEN_StdSetup(HGen, pGenData, pInp);
3. normalize the structure:
   result = STDINCHIGEN_DoNormalization(HGen, pGenData);
   optionally, look at the results;
4. obtain canonical numberings:
   result = STDINCHIGEN_DoCanonicalization(HGen, pGenData);
5. serialize, i.e. produce InChI string:
   retcode = STDINCHIGEN_DoSerialization(HGen, pGenData, pResults);
6. reset the InChI generator
   STDINCHIGEN_Reset(HGen, pGenData, pResults)
   and go to step 2 to read next structure,
   or

7. Finally destroy the generator object and free standard InChI library memories:
   STDINCHIGEN_Destroy(HGen);

Standard InChI generation API - “classic” (v. 1.01-style)

The API functions for “classic” (v. 1.01-style, non-modularized) are similar to those present in InChI software v. 1.01 and v. 1.02-beta (see, however, the notes below).

Note. Function names used in v.1.01 and v.1.02-beta have been replaced with new names containing substring “Std”. This should facilitate simultaneous use of standard and non-standard InChI software. All new names contain substring “std”.

The primary function producing standard InChI is
int INCHI_DECL GetStdINCHI(inchi_Input *inp, inchi_Output *out)

This function replaces **GetINCHI()** which was present in InChI software v. 1.01 and v. 1.02 beta.

However, the parameters of **GetStdINCHI()** (described in inchi_api.h header file) are the same as those used in **GetINCHI()** with the exception that the new function does not accept options leading to generation of non-standard InChI.

Only options affecting perception of the input structure, not InChI generation, are accepted. These are (preceded by ‘/’ under Windows or ‘-‘ under other operating systems):

NEWPSOFF	(Both ends of a wedge point to stereocenters);
DoNotAddH	(Don't add H according to usual valences: all H are explicit);
SNon	(Exclude stereo).

Also allowed are the following:

AuxNone	(Omit auxiliary information output (default: Include))
Wnumber	(Set time-out per structure in seconds; W0 means unlimited. In InChI library the default value is unlimited)
WarnOnEmptyStructure	(Warn and produce an empty InChI for an empty structure instead of an error message)

The following functions replace those available in v. 1.01:

void INCHI_DECL FreeStdINCHI (inchi_Output *out)
(replaces **FreeINCHI**)

int INCHI_DECL GetStructFromStdINCHI(inchi_InputINCHI *inpInChI, inchi_OutputStruct *outStruct)
(replaces **GetStructFromINCHI**)

void INCHI_DECL FreeStructFromStdINCHI(inchi_OutputStruct *out)
(replaces **FreeStructFromINCHI**)

int INCHI_DECL Get_std_inchi_Input_FromAuxInfo (char *szInchiAuxInfo, int bDoNotAddH, InchiInpData *pInchiInp)
(replaces **Get_inchi_Input_FromAuxInfo**)

void INCHI_DECL Free_std_inchi_Input(inchi_Input *pInp)
(replaces **Free_inchi_Input**)

The function
int INCHI_DECL GetStringLength(char *p)
remained unchanged.

Note:

The following function has been removed from standard InChI API:

int INCHI_DECL GetINCHIfromINCHI(inchi_InputINCHI *inpInChI, inchi_Output *out)

Standard InChIKey generation

The following InChI API function supports generation of standard InChIKey from standard InChI (see also inchi_api.h header file).

int GetStdINCHIKeyFromStdINCHI(const char* szINCHISource, char* szINCHIKey)

Calculate standard InChIKey from standard InChI string.
Input: szINCHISource – source null-terminated InChI string
Output: szINCHIKey - InChIKey string, null-terminated
Returns: 0 => Success, any other value is an error or warning code (see inchi_api.h)
The user-supplied buffer szINCHIKey should be at least 28 bytes long.

To check if the input szINCHISource represents a valid standard InChI identifier the beginning of string is compared with "InChI=1S/"; it is also required that the prefix is followed by at least one of a..zA..Z0..9 or '/' characters.

Note: the function **CheckINCHIKey,** which was present in InChI software v.1.02-beta has been removed because the check character is not present in the standard InChIKey.