



OpenEye
Scientific Software

LexichemTK – C++

Release 2.0.0

OpenEye Scientific Software, Inc.

November 02, 2009

CONTENTS

1	Front Matter	1
2	Theory	3
2.1	Input Name Representation	3
2.2	Output Name Representation	3
2.3	Output Name Styles	4
2.4	Examples of Name Style Differences	4
3	Examples	7
3.1	Converting Molecules to Names	7
3.2	Converting Names to Molecules	9
4	API	11
4.1	OEIUPAC Constants	11
4.2	OEIUPAC Functions	16
5	Release Notes	27
5.1	Lexichem 1.2.0	27
5.2	Lexichem 1.9	27
5.3	Lexichem 1.8	27
5.4	Lexichem 1.7	28
5.5	Lexichem 1.6	28
5.6	Lexichem 1.5	29
5.7	Lexichem 1.4	29
5.8	Lexichem 1.3	30
5.9	Lexichem 1.2	30
5.10	Lexichem 1.2	30
5.11	Lexichem 1.1	30
5.12	Lexichem 1.0	31
5.13	Lexichem 1.0.b3	31
5.14	Lexichem 1.0.b2	31
5.15	Lexichem 1.0.b1	34
5.16	Indices and tables	34
	Bibliography	35
	Index	37

FRONT MATTER

Copyright 1997-2009 OpenEye Scientific Software, Santa Fe, New Mexico. All rights reserved.

All rights reserved. This material contains proprietary information of OpenEye Scientific Software. Use of copyright notice is precautionary only and does not imply publication or disclosure.

The information supplied in this document is believed to be true but no liability is assumed for its use or the infringement of the rights of others resulting from its use. Information in this document is subject to change without notice and does not represent a commitment on the part of OpenEye Scientific Software.

This package is sold/licensed/distributed subject to the condition that it shall not, by way of trade or otherwise, be lent, re-sold, hired out or otherwise circulated without OpenEye Scientific Software's prior consent, in any form of packaging or cover other than that in which it was produced. No part of this manual or accompanying documentation, may be reproduced, stored in a retrieval system on optical or magnetic disk, tape, CD, DVD or other medium, or transmitted in any form or by any means, electronic, mechanical, photocopying recording or otherwise for any purpose other than for the purchaser's personal use without a legal agreement or other written permission granted by OpenEye.

This product should not be used in the planning, construction, maintenance, operation or use of any nuclear facility nor the flight, navigation or communication of aircraft or ground support equipment. OpenEye Scientific software, shall not be liable, in whole or in part, for any claims arising from such use, including death, bankruptcy or outbreak of war.

Windows is a registered trademark of Microsoft Corporation. Apple, OS X, and Macintosh are registered trademarks of Apple Computer, Inc. AIX and IBM are registered trademarks of International Business Machines Corporation. UNIX is a registered trademark of the Open Group. RedHat is a registered trademark of RedHat, Inc. Linux is a registered trademark of Linus Torvalds. SPARC is a registered trademark of SPARC International Inc.

SYBYL is a registered trademark of TRIPOS, Inc. MDL is a registered trademark and ISIS is a trademark of MDL Information Systems, Inc. SMILES, SMARTS, and SMIRKS may be trademarks of Daylight Chemical Information Systems. Macromodel is a trademark of Schrodinger, Inc.

Python is a trademark of the Python Software Foundation. Java is a trademark or registered trademark of Sun Microsystems, Inc. in the U.S. and other countries.

Other products and software packages referenced in this document are trademarks and registered trademarks of their respective vendors or manufacturers.

THEORY

2.1 Input Name Representation

The *oeiupac* library currently processes NUL (zero) terminated ASCII character strings, therefore Greek characters, symbols, fonts and superscripts must be transliterated into the printable subset of ASCII. When parsing compound names, the *oeiupac* library considers both spaces and tab characters as interchangeable, and any number of consecutive 'whitespace' characters are treated as a single space.

Currently, the name parsing is case insensitive, allowing arbitrary mixing of upper and lower case characters, *e.g.* initial letter capitalization.

Greek characters are understood in a number of different representations. The strings '\$a', '\${a}', 'alpha', '.alpha.', 'α', 'α' and 'α' are all understood to represent the Greek character *alpha*, (α).

There is no special representation for italic characters. Compound names such as '*tert*-butyl' and '*p*-aminobenzamidine' are represented as 'tert-butyl' and 'p-aminobenzamidine'. Both the long and short forms of prefixes can be used, allowing the above examples to also be written as 't-butyl' and 'para-aminobenzamidine'.

2.2 Output Name Representation

Unrecognized functional groups, linkers or ring systems are denoted in the generated name as the string '**BLAH**'. As much of the name possible is generated resulting in compound names such as 'dichloroBLAHcarboxylic acid'. Generated compound names are entirely lower case, with no initial capitalization. Upper case characters are generated for locants and as described above, for BLAH.

When generating Greek characters in compound names, the *oeiupac* library currently uses the dollar character followed by single letter representation. In this formalism, '\$a' represents the Greek character alpha, α , '\$b' the Greek character beta, β , '\$g' the Greek character gamma, γ and '\$l' the Greek character lambda, λ .

When generating superscripts, the *oeiupac* library currently uses the caret and curly braces representation. Hence '\$l^{5}' represents the Greek character lambda followed by a superscript five, *i.e.* λ^5 . Similarly, 'pentacyclo[4.2.0.0^{2,5}.0^{3,8}.0^{4,7}]octane' would be the von Baeyer system name for cubane, *i.e.*

pentacyclo [4.2.0.0^{2,5}.0^{3,8}.0^{4,7}] octane.

Multiple components in a disconnected molecule, apart from common salts and counter ions, are separated from each other by a semicolon followed by a space. Mixtures containing salts are written ordering the cations, before the compound name, followed by anions, finally followed by any common neutral molecules (*e.g.* hydrate or hydrochloride).

2.3 Output Name Styles

The *Lexichem* compound naming functionality supports the generation of several *styles* of compound name. The currently predefined name styles are *OpenEye* (the default), IUPAC, CAS, Traditional and Systematic. *OpenEye* names loosely correspond to the kinds of names familiar to a medicinal chemist. These names are intended to be a subset of the IUPAC 2005 standard's acceptable names, but not necessarily the PIN (Preferred IUPAC Name). These correspond to the types of names found in a Sigma-Aldrich catalog or a Journal of Medicinal Chemistry article for example.

IUPAC names are intended to follow the IUPAC 2005 recommendations for the Preferred IUPAC Name (PIN). Unfortunately, this functionality is relatively recent, so the best that can be hoped for these names is that they are more IUPAC-like than the default *OpenEye* name style. Future release of *Lexichem* may further refine this definition to provide IUPAC2005, IUPAC93 and IUPAC79 name styles that reflect the corresponding standard's preferred name.

The *Lexichem* CAS name style is intended to follow the Chemical Abstracts Service's naming conventions, where they differ from IUPAC's. Once again, as this functionality is relatively recent, the effect is to generate names that are more CAS-like than the default *OpenEye* name style.

The Traditional name style corresponds to forms of compound naming that are now no longer acceptable to the IUPAC rules. The boundary between whether a trivial/common name is considered *OpenEye* or Traditional when it acceptable to IUPAC but not preferred is blurred, with *OpenEye* attempting to follow the more prevalent usage.

Finally, Systematic names correspond to the fully systematic IUPAC names that the IUPAC preferred names are slowly converging towards.

2.4 Examples of Name Style Differences

Some of the concepts explained in the previous section are probably best clarified through some real examples.

2.4.1 Example OpenEye vs. IUPAC vs. Systematic Differences

The SMILES string O is called 'water' by the *OpenEye* name style, but 'oxidane' by the IUPAC and Systematic name styles.

The SMILES C#C is called 'acetylene' by the *OpenEye* and IUPAC name styles, but 'ethyne' by the Systematic name style.

The SMILES prefix *Nc1ccccc1 is called 'anilino' by *OpenEye* and IUPAC, but 'phenylamino' by systematic.

The SMILES prefix *O[N+]#[C-] is called 'fulminato' by *OpenEye*, but 'isocyanooxy' by IUPAC and Systematic.

The SMILES prefix *C(=O)C is called 'acetyl' in *OpenEye* and IUPAC, but 'ethanoyl' in Systematic.

The SMILES string CC(=O)C is called 'acetone' in *OpenEye*, but 'propan-2-one' in IUPAC and Systematic.

The SMILES string C12C3C4C1C5C4C3C25 is called 'cubane' in *OpenEye*, but is currently named 'BLAH' in IUPAC and Systematic as we currently fail to name it as the preferred IUPAC2005 PIN: 'pentacyclo[4.2.0.0^{2,5}.0^{3,8}.0^{4,7}]octane'.

The SMILES string C(=O)O is called 'formic acid' in *OpenEye*/IUPAC, but 'methanoic acid' in Systematic.

2.4.2 Example OpenEye/IUPAC vs. CAS Differences

The SMILES string c1ccccc1CCCCCCC is named as '1-phenylheptane' by *OpenEye* and IUPAC, but as 'heptylbenzene' by CAS.

The SMILES prefix *[BH2] is called 'boranyl' by *OpenEye* and IUPAC, but as 'boryl' by CAS.

2.4.3 Example OpenEye/IUPAC vs. Traditional Differences

The SMILES prefix `*S` is called ‘sulfanyl’ by *OpenEye* and IUPAC, but as ‘mercapto’ by Traditional.

The SMILES string `CCCCCCCCC(=O)O` is called ‘nonanoic acid’ by *OpenEye* and IUPAC, but as ‘pelargonic acid’ by Traditional.

EXAMPLES

3.1 Converting Molecules to Names

Listing 1: Converting molecules to names

```
/*  
Copyright (C) 2003, 2004, 2005, 2006, 2007, 2008, 2009  
OpenEye Scientific Software, Inc.  
*/  
#include "openeye.h"  
  
#include <stdlib.h>  
#include <string>  
  
#include "oeplatform.h"  
#include "oesystem.h"  
#include "oechem.h"  
#include "oeiupac.h"  
  
#include "mol2nam_example.itf"  
  
using namespace OEPlatform;  
using namespace OESystem;  
using namespace OEChem;  
using namespace OEIUPAC;  
using namespace std;  
  
int main(int argc, char *argv[])  
{  
    OESetMemPoolMode(OEMemPoolMode::SingleThreaded|  
                    OEMemPoolMode::UnboundedCache);  
  
    OEThrow.Info("Lexichem mol2nam example");  
    OEThrow.Info("  Lexichem version: %s", OEIUPACGetRelease());  
  
    OEInterface itf(InterfaceData, argc, argv);  
  
    unsigned int language=OEGetIUPACLanguage(itf.Get<string>("-language"));  
    unsigned int charset=OEGetIUPACCharSet(itf.Get<string>("-encoding"));  
    const unsigned char *style=OEGetIUPACNamStyle(itf.Get<string>("-style"));  
  
    oemolistream ifs(itf.Get<string>("-in"));
```

```
if (!ifs)
    OEThrow.Fatal("Unable to open %s for reading",
                 itf.Get<string>("-in").c_str());

oemolostream ofs;
string outname="";
if (itf.Has<string>("-out"))
{
    outname=itf.Get<string>("-out");
    if (!ofs.open(outname))
        OEThrow.Fatal("Unable to open %s for writing", outname.c_str());
}

OEGraphMol mol;
std::string tmp;
while (OEReadMolecule(ifs, mol))
{
    std::string name = OEIUPAC::OECreatIUPACName(mol, style);

    if (language)
        name = OEIUPAC::OEToLanguage(name.c_str(), language);
    if (itf.Get<bool>("-capitalize"))
        name = OEIUPAC::OECapitalizeName(name.c_str());

    switch (charset)
    {
    case OECharSet::ASCII:
        name = OEIUPAC::OEToASCII(name.c_str());
        break;
    case OECharSet::UTF8:
        name = OEIUPAC::OEToUTF8(name.c_str());
        break;
    case OECharSet::HTML:
        name = OEIUPAC::OEToHTML(name.c_str());
        break;
    case OECharSet::SJIS:
        name = OEIUPAC::OEToSJIS(name.c_str());
        break;
    case OECharSet::EUCJP:
        name = OEIUPAC::OEToEUCJP(name.c_str());
        break;
    }

    if (outname.size()>0)
    {
        if (itf.Has<string>("-delim"))
        {
            const char *title = mol.GetTitle();
            if(title && *title)
            {
                tmp = name;
                name = title;
                name.append(itf.Get<string>("-delim"));
                name.append(tmp);
            }
        }

        if (itf.Has<string>("-tag"))
```

```

        OESetSDDData(mol, itf.Get<string>("-tag"), name);

        mol.SetTitle(name);
        OEWriteMolecule(ofs, mol);
    }
    else printf("%s\n", name.c_str());
}

return 0;
}

```

3.2 Converting Names to Molecules

Listing 2: Converting names to molecules

```

/*****
Copyright (C) 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009
OpenEye Scientific Software, Inc.
*****/
#include "openeye.h"

#include "oeplatform.h"
#include "oesystem.h"
#include "oechem.h"
#include "oeiupac.h"

#include "nam2mol_example.itf"

using namespace OEPlatform;
using namespace OESystem;
using namespace OEChem;
using namespace OEIUPAC;
using namespace std;

#ifdef STDIN_FILENO
#define STDIN_FILENO 0
#endif

int main(int argc, char *argv[])
{
    OESetMemPoolMode(OEMemPoolMode::SingleThreaded|OEMemPoolMode::UnboundedCache);

    OEThrow.Info("Lexichem nam2mol example");
    OEThrow.Info("  Lexichem version: %s", OEIUPACGetRelease());

    OEInterface itf(InterfaceData, argc, argv);

    oeifstream infile;
    string inname=itf.Get<string>("-in");
    if (inname=="-")
    {
        if (!infile.openfd(STDIN_FILENO, true)) // read from stdin
            OEThrow.Fatal("Unable to read from stdin");
    }
}

```

```
else
{
    if (!infile.open(inname))
        OThrow.Fatal("Unable to open input file: %s\n", inname.c_str());
}

oemolostream outfile;
if (!outfile.open(itf.Get<string>("-out")))
    OThrow.Fatal("Unable to create output file: %s\n",
        itf.Get<string>("-out").c_str());

unsigned int language = OEGetIUPACLanguage(itf.Get<string>("-language"));

OEGraphMol mol;
char buffer[8192];
bool done;

while (infile.getline(buffer, 8192))
{
    mol.Clear();

    // Speculatively reorder CAS permuted index names
    std::string str = OEReorderIndexName(buffer);
    if (str.empty()) str = buffer;

    if (language != OELanguage::AMERICAN)
    {
        str = OEFromUTF8(str.c_str());
        str = OELowerCaseName(str.c_str());
        str = OEFromLanguage(str.c_str(), language);
    }

    done = OEParseIUPACName(mol, str.c_str());

    if (!done && itf.Get<bool>("-empty"))
    {
        mol.Clear();
        done = true;
    }
    if (done)
    {
        if (itf.Has<string>("-tag"))
            OESetSSData(mol, itf.Get<string>("-tag"), buffer);
        mol.SetTitle(buffer);
        OEWriteMolecule(outfile, mol);
    }
}

return 0;
}
```

4.1 OEIUPAC Constants

4.1.1 OEIUPAC

This namespace contains constants.

OENamStyleOpenEye

This constant is used to specify the (default) predefined OpenEye name style to the `OECreatIUPACName` function.

OENamStyleTraditional

This constant is used to specify the predefined traditional name style to the `OECreatIUPACName` function.

OENamStyleSystematic

This constant is used to specify the predefined systematic name style to the `OECreatIUPACName` function.

OENamStyleIUPAC

This constant is used to specify the predefined IUPAC 200x name style to the `OECreatIUPACName` function.

OENamStyleCAS

This constant is used to specify the predefined Chemical Abstracts Service (CAS) name style to the `OECreatIUPACName` function.

OENamStyleCASIndex

This constant is used to specify the predefined Chemical Abstracts Service (CAS) permuted index name style to the `OECreatIUPACName` function.

OENamStyleAutoNom

This constant is used to specify the predefined MDL/Beilstein AutoNom name style to the `OECreatIUPACName` function.

OENamStyleIUPAC79

This constant is used to specify the predefined IUPAC 1979 name style to the `OECreatIUPACName` function.

OENamStyleIUPAC93

This constant is used to specify the predefined IUPAC 1993 name style to the `OECreatIUPACName` function.

OENamStyleACDName

OE_LANG_ENGLISH

This constant is used to specify the American dialect of English to the `OEFromLanguage` and the `OEToLanguage` function. This is currently a synonym of the constant `OEIUPAC::OE_LANG_AMERICAN`.

OE_LANG_AMERICAN

This constant is used to specify the American dialect of English to the `OEFromLanguage` and the `OEToLanguage` function. This is currently a synonym of the constant `OEIUPAC::OE_LANG_ENGLISH`.

OE_LANG_BRITISH

This constant is used to specify the traditional British dialect of English to the `OEFromLanguage` and the `OEToLanguage` function.

OE_LANG_INTERNATIONAL

This constant is used to specify the international dialect of English to the `OEToLanguage` function.

OE_LANG_CHINESE

This constant is used to specify the Chinese language to the `OEFromLanguage` and the `OEToLanguage` functions.

OE_LANG_DANISH

This constant is used to specify the Danish language to the `OEFromLanguage` and the `OEToLanguage` functions.

OE_LANG_DUTCH

This constant is used to specify the Dutch language to the `OEFromLanguage` and the `OEToLanguage` functions.

OE_LANG_FRENCH

This constant is used to specify the French language to the `OEFromLanguage` and the `OEToLanguage` functions.

OE_LANG_GERMAN

This constant is used to specify the German language to the `OEFromLanguage` and the `OEToLanguage` functions.

OE_LANG_GREEK

This constant is used to specify the Greek language to the `OEFromLanguage` and the `OEToLanguage` functions.

OE_LANG_HUNGARIAN

This constant is used to specify the Hungarian language to the `OEFromLanguage` and the `OEToLanguage` functions.

OE_LANG_IRISH

This constant is used to specify the Irish language to the `OEFromLanguage` and the `OEToLanguage` functions.

OE_LANG_ITALIAN

This constant is used to specify the Italian language to the `OEFromLanguage` and the `OEToLanguage` functions.

OE_LANG_JAPANESE

This constant is used to specify the Japanese language to the `OEFromLanguage` and the `OEToLanguage` functions.

OE_LANG_POLISH

This constant is used to specify the Polish language to the `OEFromLanguage` and the `OEToLanguage` functions.

OE_LANG_PORTUGUESE

This constant is used to specify the Portuguese language to the `OEFromLanguage` and the `OEToLanguage` functions.

OE_LANG_ROMANIAN

This constant is used to specify the Romanian language to the `OEFromLanguage` and the `OEToLanguage` functions.

OE_LANG_RUSSIAN

This constant is used to specify the Russian language to the `OEFromLanguage` and the `OEToLanguage` functions.

OE_LANG_SLOVAK

This constant is used to specify the Slovak language to the `OEFromLanguage` and the `OEToLanguage` functions.

OE_LANG_SPANISH

This constant is used to specify the Spanish language to the `OEFromLanguage` and the `OEToLanguage` functions.

OE_LANG_SWEDISH

This constant is used to specify the Swedish language to the `OEFromLanguage` and the `OEToLanguage` functions.

OE_LANG_WELSH

This constant is used to specify the Welsh language to the `OEFromLanguage` and the `OEToLanguage` functions.

4.1.2 OELanguage

This namespace contains constants for defining which language to translate to/from using `OEFromLanguage` and `OEToLanguage`.

ENGLISH

AMERICAN

BRITISH

INTERNATIONAL

CHINESE

DANISH

DUTCH

FRENCH

GERMAN

GREEK

HUNGARIAN

IRISH

ITALIAN

JAPANESE

POLISH

PORTUGUESE

ROMANIAN

RUSSIAN

SLOVAK

SPANISH

SWEDISH

WELSH

4.2 OEIUPAC Functions

```
std::string OECapitalizeName(const char *ptr)
```

Capitalize the appropriate first letter of a name generated by `OECreatelUPACName`. This function should be called after translating the name to a foreign language, but before converting the character set encoding.

4.2.2 OECreatelUPACName

```
std::string OECreatelUPACName(const OEChem::OEMolBase &mol,
                             const unsigned char *style=OENamStyleOpenEye)
```

This function attempts to generate a ‘reasonable’ IUPAC name for the given molecule, mol, and return the result in a C++ STL string. These ‘reasonable’ names attempts to be one of the recommended IUPAC names for a compound, however occasionally this function may fall back to using IUPAC ‘systematic’ naming for parts of a molecule. Any parts of a molecule that cannot be named, result in the substring BLAH appearing in the returned string.

The optional style argument can be used to control and customize the style of the names generated by this function. The nine currently predefined name styles are `OEIUPAC::OENamStyleOpenEye` (the default), `OEIUPAC::OENamStyleIUPAC`, `OEIUPAC::OENamStyleIUPAC79`, `OEIUPAC::OENamStyleIUPAC93`, `OEIUPAC::OENamStyleTraditional`, `OEIUPAC::OENamStyleAutoNom`, `OEIUPAC::OENamStyleCAS`, `OEIUPAC::OENamStyleCASIndex` and `OEIUPAC::OENamStyleSystematic`.

After the name has been generated it may be translated into one of several languages, for example using the `OEToGerman` or `OEToJapanese` functions, then optionally capitalized using `OECapitalizeName`, and finally converted into a final character encoding, for example using `OEToUTF8` or `OEToHTML`.

4.2.3 OEFromChinese

```
std::string OEFromChinese(const char *ptr)
```

Convert the chemical name specified by ‘ptr’ from simplified or traditional Chinese to English.

4.2.4 OEFromDanish

```
std::string OEFromDanish(const char *ptr)
```

Convert the chemical name specified by ‘ptr’ from Danish to English.

4.2.5 OEFromDutch

```
std::string OEFromDutch(const char *ptr)
```

Convert the chemical name specified by ‘ptr’ from Dutch to English.

4.2.6 OEFromFrench

```
std::string OEFromFrench(const char *ptr)
```

Convert the chemical name specified by 'ptr' from French to English.

4.2.7 OEFromGerman

```
std::string OEFromGerman(const char *ptr)
```

Convert the chemical name specified by 'ptr' from German to English.

4.2.8 OEFromGreek

```
std::string OEFromGreek(const char *ptr)
```

Convert the chemical name specified by 'ptr' from Greek to English.

4.2.9 OEFromHungarian

```
std::string OEFromHungarian(const char *ptr)
```

Convert the chemical name specified by 'ptr' from Hungarian to English.

4.2.10 OEFromIrish

```
std::string OEFromIrish(const char *ptr)
```

Convert the chemical name specified by 'ptr' from Irish to English.

4.2.11 OEFromItalian

```
std::string OEFromItalian(const char *ptr)
```

Convert the chemical name specified by 'ptr' from Italian to English.

4.2.12 OEFromJapanese

```
std::string OEFromJapanese(const char *ptr)
```

Convert the chemical name specified by 'ptr' from Japanese to English.

4.2.13 OEFromKOI8R

```
std::string OEFromKOI8R(const char *ptr)
```

Convert the string 'ptr' from the Russian KOI-8 character encoding to *Lexichem*'s default encoding using `\u` escapes to represent non-ASCII unicode characters.

4.2.14 OEFromLanguage

```
std::string OEFromLanguage(const char *ptr, unsigned int lang)
```

This is a helper wrapper function that can be used to convert the chemical name specified by 'ptr' from the language specified by 'lang' to English. The languages are specified by the constants with the `OE_LANG_` prefix described in the `OEIUPAC` namespace.

4.2.15 OEFromPolish

```
std::string OEFromPolish(const char *ptr)
```

Convert the chemical name specified by 'ptr' from Polish to English.

4.2.16 OEFromRomanian

```
std::string OEFromRomanian(const char *ptr)
```

Convert the chemical name specified by 'ptr' from Romanian to English.

4.2.17 OEFromRussian

```
std::string OEFromRussian(const char *ptr)
```

Convert the chemical name specified by 'ptr' from Russian to English.

4.2.18 OEFromSlovak

```
std::string OEFromSlovak(const char *ptr)
```

Convert the chemical name specified by 'ptr' from Slovak to English.

4.2.19 OEFromSpanish

```
std::string OEFromSpanish(const char *ptr)
```

Convert the chemical name specified by 'ptr' from Spanish to English.

4.2.20 OEFromSwedish

```
std::string OEFromSwedish(const char *ptr)
```

Convert the chemical name specified by 'ptr' from Swedish to English.

4.2.21 OEFromUTF8

```
std::string OEFromUTF8(const char *ptr)
```

Convert the string 'ptr' from the UTF-8 character encoding to *Lexichem*'s default encoding using 'u' escapes to represent non-ASCII unicode characters.

4.2.22 OEFromWelsh

```
std::string OEFromWelsh(const char *ptr)
```

Convert the chemical name specified by 'ptr' from simplified or traditional Welsh to English.

4.2.23 OEGetCIPStereo

```
char OEGetCIPStereo(const OEMolBase &mol,  
                   const OEAtomBase *atm)  
char OEGetCIPStereo(const OEMolBase &mol,  
                   const OEBondBase *bnd)
```

These functions return the Cahn-Ingold-Prelog descriptor for the given atom or bond stereo center, from the stereochemistry set on the `OEMolBase`. For chiral atom centers, the `OEAtomBase` form of this function returns either 'R' or 'S' for specified CIP stereo centers, 'N' for CIP stereo centers that don't have stereo specified (*i.e.* `OEAtomBase::HasStereoSpecified` returns false), and 'X' for atoms that are not CIP stereo centers. For double bonds, the `OEBondBase` form of this function returns either 'E' or 'Z' specified CIP stereo centers, 'N' for CIP stereo centers that don't have stereo specified (*i.e.* `OEBondBase::HasStereoSpecified` returns false), and 'X' for bonds that are not CIP stereo centers.

4.2.24 OEIUPACGetArch

```
const char *OEIUPACGetArch()
```

4.2.25 OEIUPACGetLicensee

```
bool OEIUPACGetLicensee(std::string &licensee)
```

4.2.26 OEIUPACGetPlatform

```
const char *OEIUPACGetPlatform()
```

4.2.27 OEIUPACGetRelease

```
const char *OEIUPACGetRelease()
```

4.2.28 OEIUPACGetSite

```
bool OEIUPACGetSite(std::string &site)
```

4.2.29 OEIUPACGetVersion

```
unsigned int OEIUPACGetVersion()
```

4.2.30 OEIUPACIsLicensed

```
bool OEIUPACIsLicensed(const char *feature=0, unsigned int *expdate=0)
```

Determine whether a valid license file is present. This function may be called without a legitimate run-time license to determine whether it is safe to call any of OEIUPAC's functionality.

The 'features' argument can be used to check for a valid license to OEIUPAC along with that feature. For example, to verify that OEIUPAC can be used from Python:

```
if (!OEIUPACIsLicensed("python"))
    OThrow.Warning("OEIUPAC is not licensed for the python feature");
```

The second argument can be used to get the expiration date of the license. This is an array of size three with the date returned as {day, month, year}. Even if the function returns `false` due to an expired license, the `expdate` will show that expiration date. A value of a zeroes implies that no license or date was found.

```
unsigned int expdate[3];
if (OEIUPACIsLicensed(0, expdate))
{
    OThrow.Info("License expires: day: %d month: %d year: %d",
               expdate[0], expdate[1], expdate[2]);
}
```

4.2.31 OELowerCaseName

```
std::string OELowerCaseName(const char *ptr)
```

Convert the specified compound name to lower-case. This function understands chemical nomenclature and foreign alphabets. This function should be called prior to translating a name from a foreign language, as the language translation routines assume that the name is lower-case.

4.2.32 OENAMELocant

```
std::string OENAMELocant(unsigned int loc, bool iupac=false)
```

Generate the lexical form of a *LexiChem* locant index. The molecule's created by the function `OEParseIUPACName` are annotated with locant information in each atom's integer atom type field. These locant indices may be retrieved using `OEChem's OeAtomBase::GetIntType` method.

4.2.33 OEParseIUPACName

```
bool OEParseIUPACName(OEChem::OEMolBase &mol, const char *name)
```

This function parses the compound name (not necessarily a systematic or preferred IUPAC name) from the NUL-terminated string given by name, and places the processed molecule in mol. This function returns `true` if the name could be parsed without problems. When returning `false`, the contents of mol contain as much of the name as could be processed.

4.2.34 OEReorderIndexName

```
std::string OEReorderIndexName(const char *ptr)
```

Attempt to reorder the specified permuted index name. The permute strings such as “benzene, chloro-” into the form “chloro-benzene” which can then be processed by *Lexichem's* `OEParseIUPACName` function.

This function returns an empty string (or the original input string) if it's argument is not recognized as a permuted index name.

4.2.35 OESetCIPStereo

```
bool OESetCIPStereo(OEChem::OEMolBase &mol, OEChem::OeAtomBase *atm, char s)
bool OESetCIPStereo(OEChem::OEMolBase &mol, OEChem::OeBondBase *bnd, char s)
```

These functions set the internal `OEChem` stereochemistry from the given CIP stereo descriptor. For the `OeAtomBase` form, the descriptor `s` must be either 'R' or 'S', and for the `OeBondBase` form, the descriptor `s` must be either 'E' or 'Z'. This function returns `true` if the stereochemistry was successfully set, and `false` otherwise: *i.e.* the descriptor was invalid or the specified atom or bond was not a CIP stereo center.

4.2.36 OETOASCII

```
std::string OETOASCII(const char *ptr)
```

Convert the string 'ptr', typically the output of the function `OECREATEIUPACName` post-processed by a language translation function, from the default ISO-8859-1 encoding, which includes 8-bit European characters, into a reduced 7-bit ASCII representation by stripping the accents off of the 8-bit characters.

4.2.37 OEToBritish

```
std::string OEToBritish(const char *ptr, bool sulph)
```

Convert the string 'ptr', typically the output of the function `OECreatIUPACName`, from the default American output, to either British spelling (when 'sulph' is true) or IUPAC international spelling (when 'sulph' is false).

4.2.38 OEToChinese

```
std::string OEToChinese(const char *ptr)
```

Convert the string 'ptr', typically the output of the function `OECreatIUPACName`, from English to Simplified Chinese.

4.2.39 OEToDanish

```
std::string OEToDanish(const char *ptr)
```

4.2.40 OEToDutch

```
std::string OEToDutch(const char *ptr)
```

Convert the string 'ptr', typically the output of the function `OECreatIUPACName`, from English to Dutch.

4.2.41 OEToEUCJP

```
std::string OEToEUCJP(const char *ptr)
```

Convert the string 'ptr', typically the output of the function `OECreatIUPACName` post-processed by a call to `OEToJapanese`, from the default encoding which uses 'u' escapes to represent unicode characters to instead use the EUC-JP character encoding for japanese characters.

4.2.42 OEToFrench

```
std::string OEToFrench(const char *ptr)
```

Convert the string 'ptr', typically the output of the function `OECreatIUPACName`, from English to French.

4.2.43 OEToGerman

```
std::string OEToGerman(const char *ptr)
```

Convert the string 'ptr', typically the output of the function `OECreatIUPACName`, from English to German.

4.2.44 OEToGreek

```
std::string OEToGreek(const char *ptr)
```

4.2.45 OEToHTML

```
std::string OEToHTML(const char *ptr)
```

Convert the string 'ptr', typically the output of the function `OECreatIUPACName` possibly post-processed by a language translation function, from the default encoding to use HTML mark-up to represent accented characters, unicode characters and superscripts.

4.2.46 OEToHungarian

```
std::string OEToHungarian(const char *ptr)
```

Convert the string 'ptr', typically the output of the function `OECreatIUPACName`, from English to Hungarian.

4.2.47 OEToIrish

```
std::string OEToIrish(const char *ptr)
```

Convert the string 'ptr', typically the output of the function `OECreatIUPACName`, from English to Irish.

4.2.48 OEToItalian

```
std::string OEToItalian(const char *ptr)
```

Convert the string 'ptr', typically the output of the function `OECreatIUPACName`, from English to Italian.

4.2.49 OEToJapanese

```
std::string OEToJapanese(const char *ptr)
```

Convert the string 'ptr', typically the output of the function `OECreatIUPACName`, from English to Japanese.

4.2.50 OEToLanguage

```
std::string OEToLanguage(const char *ptr, unsigned int lang)
```

This is a helper wrapper function that can be used to convert the chemical name specified by 'ptr' from English to the language specified by 'lang'. The languages are specified by the constants with the `OE_LANG_` prefix described in the `OEIUPAC` namespace.

4.2.51 OEToPolish

```
std::string OEToPolish(const char *ptr)
```

Convert the string 'ptr', typically the output of the function `OECreatIUPACName`, from English to Polish.

4.2.52 OEToRomanian

```
std::string OEToRomanian(const char *ptr)
```

Convert the string 'ptr', typically the output of the function `OECreatIUPACName`, from English to Romanian.

4.2.53 OEToRussian

```
std::string OEToRussian(const char *ptr)
```

Convert the string 'ptr', typically the output of the function `OECreatIUPACName`, from English to Russian.

4.2.54 OEToSJIS

```
std::string OEToSJIS(const char *ptr)
```

Convert the string 'ptr', typically the output of the function `OECreatIUPACName` post-processed by a call to `OEToJapanese`, from the default encoding which uses 'u' escapes to represent unicode characters to instead use the Shift-JIS character encoding for japanese characters.

4.2.55 OEToSlovak

```
std::string OEToSlovak(const char *ptr)
```

Convert the string 'ptr', typically the output of the function `OECreatIUPACName`, from English to Slovak.

4.2.56 OEToSpanish

```
std::string OEToSpanish(const char *ptr)
```

Convert the string 'ptr', typically the output of the function `OECreatIUPACName`, from English to Spanish.

4.2.57 OEToSwedish

```
std::string OEToSwedish(const char *ptr)
```

Convert the string 'ptr', typically the output of the function `OECreatIUPACName`, from English to Swedish.

4.2.58 OEToUTF8

```
std::string OEToUTF8(const char *ptr)
```

Convert the string 'ptr', typically the output of the function `OECreatIUPACName` post-processed by a language translation function, from the default ISO-8859-1 encoding which uses 'u' escapes to represent unicode characters to instead use the UTF-8 character encoding for accented and other characters.

4.2.59 OEToWelsh

```
std::string OEToWelsh(const char *ptr)
```

Convert the string 'ptr', typically the output of the function `OECreatIUPACName`, from English to Welsh.

RELEASE NOTES

5.1 Lexichem 1.2.0

- This release includes the ability to parse stereo on input names. Previously it was read and ignored.
- Fixed a bug where, in rare cases, the output name depended on input atom ordering.
- Fixed a crash in determining CIP stereo for very large, pathological molecules.

5.2 Lexichem 1.9

- On a benchmark of 250251 compounds in the NCI00 database, `mol2nam` is able to convert 234297 structures (93.62%) to names without BLAH. Of these 234297 names, `nam2mol` is able to convert 231566 (98.83%) back into structures.
- This release includes a significant number of improvements to both name generation and name parsing. Several bugs have also been fixed. The name parsing conversion rate for the 71367 compound names in the 2003 Maybridge catalog is now up to 95.24%.
- Several improvements have been made to the specification of CIP stereochemistry during name generation. For example, previously linking groups such as `amidino`, `carbamimidoyl` and `diazenyl` would forget to specify E/Z descriptors if they contained a chiral double bond with specified stereochemistry. We would also fail to place some chiral prefixes such as `(E)-styr1` and `(Z)-cinnamyl` in brackets which can lead to ambiguity when interpreting the generated name.

5.3 Lexichem 1.8

- On a benchmark of 250251 compounds in the NCI00 database, `mol2nam` is able to convert 234296 structures (93.62%) to names without BLAH. Of these 234296 names, `nam2mol` is able to convert 228102 (97.36%) back into structures.
- This release includes a significant number of improvements to both name generation and name parsing. Several bugs have also been fixed. The name parsing conversion rate for the 71367 compound names in the 2003 Maybridge catalog is now up to 95.12%.
- One of the major parsing improvements in this release is the much improved support for handling von Baeyer ring nomenclature. We can now parse names such as `'1,4-dithioniabicyclo[2.2.2]octane'`, `'bicyclo[4.2.0]octa-1(6),2,4-triene'` and `'2,4-diazaspiro[4.4]nonane-1,3-dione'`.

5.4 Lexichem 1.7

- On a benchmark of 250251 compounds in the NCI00 database, `mol2nam` is able to convert 234155 structures (93.57%) to names without BLAH. Of these 234155 names, `nam2mol` is able to convert 223246 (95.34%) back into structures.
- This release includes a significant number of improvements to both name generation and name parsing. Several bugs have also been fixed. The name parsing conversion rate for the 71367 compound names in the 2003 Maybridge catalog is now up to 93.81%.
- A new `OELowerCaseName` function has been added to the *Lexichem* toolkit API. This function converts the input chemical name to lower-case, whilst preserving the case sensitive aspects of IUPAC names. This functionality allows uppercase and mixed case names to be translated into English, as the `OEFrom<Foo>` functions assume their input is lowercase. For example, this feature allows AGUA to be recognized via `OEFromSpanish`.
- A new `OEReorderIndexName` function has been added to the *Lexichem* toolkit API. This function attempts to reorder the given permuted index name into a form that can be handled by the `OEParseIUPACName` function. For example, this will convert the string ‘benzene, chloro-’ into ‘chloro-benzene’.
- A number of improvements and bug fixes have been made to *Lexichem*’s naming styles. For example, AutoNom and CAS permuted index styles are now far more AutoNom-like and CAS-like respectively. Naming of metallocenes and fullerenes is much improved.
- Some dramatic improvements have been made with foreign language support. On the 250251 compounds in the NCI00 database mentioned above, we now round-trip 100% to German and back without any differences. Japanese, Spanish and Swedish rates are all currently above 99%. Support for Hungarian and Polish has been dramatically improved.

5.5 Lexichem 1.6

- On a benchmark of 250251 compounds in the NCI00 database, `mol2nam` is able to convert 233010 structures (93.11%) to names without BLAH. Of these 233010 names, `nam2mol` is able to convert 221331 (94.99%) back into structures.
- This release includes a significant number of improvements to both name generation and name parsing. For example, both name generation and parsing now do a much better job on ring fusion nomenclature, for names like ‘5,6,7,8-tetrahydro[1,2,4]triazolo[4,3-a]pyridine’. There’s also much improved handling of charged ring systems. The name parsing conversion rate for the 71367 compound names in the 2003 Maybridge catalog is now 93.25% in v1.6, up from 80.80% in v1.5.
- In name generation, new naming styles have been added for MDL/Beilstein AutoNom style names, for CAS permuted index style names (and there are new placeholder styles for IUPAC79 and IUPAC93 naming). A large number of improvements have been made to names generated using the ‘traditional’ naming style. A new `OECapitalizeName` API function is available to capitalizing the appropriate first letter of a generated name, such as ‘p-tert-Butylbenzoic acid’.
- Several bug fixes have been made to the Cahn-Ingold-Prelog (CIP) chirality perception implementation.
- The `OEParseIUPACName` function is now able return supplementary locant annotations for each atom. This function now stores an integer locant code/identifier in the integer atom type field of each atom, which may be retrieved using the `OEAtomBase::GetIntType` method and converted into a readable/displayable string using the recently exposed `OENAMELocant` function. This functionality is a recent addition (obviously), and most but not all supported ring systems and parents have locant annotations in this initial release.

- Finally, for the adventurous, new APIs for translating compound names from foreign languages into English are available as the experimental `OEFFromJapanese`, `OEFFromSwedish` and `OEFFromSpanish` functions. Additionally, a `OEFFromUTF8` function is available for converting UTF-8 encoded strings into the escaped sequences expected by these functions (effectively the inverse of `OEToUTF8`).

5.6 Lexichem 1.5

- On a benchmark of 250251 compounds in the NCI00 database, `mol2nam` is able to convert 223066 structures (89.14%) to names without BLAH. Of these 223066 names, `nam2mol` is able to convert 192487 (86.29%) back into structures.
- This release includes a significant number of improvements to both name generation and name parsing. For example, `nam2mol` now supports more numbered locants, such as ‘N1-methylaniline’ and for ‘Maybridge-style’ locant names such as N’ 1 (interpreted as the more common N1’). These and similar changes have increased the conversion rate for the 71367 compound names in the 2003 Maybridge catalog, from 69.51% in v1.4 to 80.80% in v1.5.
- This release includes the ability to generate compound names in Japanese, and much improved Spanish and Polish naming support. In order to better support internationalization, APIs are now available to map from the default ISO-8859-1 output to either 7-bit ASCII, UTF-8, HTML and for Japanese locales, Shift-JIS or EUC-JP.
- Although impossible in the general case, several improvements have been made to *Lexichem*’s compound naming such that the assigned names are now more stable under arbitrary input ordering of atoms and bonds.

5.7 Lexichem 1.4

- On a benchmark of 250251 compounds in the NCI00 database, `mol2nam` is able to convert 221254 structures (88.41%) to names without BLAH. Of these 221254 names, `nam2mol` is able to convert 192345 (86.93%) back into structures.
- *Lexichem* v1.4 is predominantly a maintenance to provide a version of the *oeiupac* library that is compatible with *OEChem* v1.4. However, there have been a number of significant improvements to name parsing, and minor improvements to name generation since last month’s v1.3 release.
- This release also includes the ability to generate compound names in several languages. In addition, to British spellings, *Lexichem* can now generate German, Italian, French, Spanish, Swedish, Dutch and Polish names. Whilst the translations for German, Italian, Swedish and Polish are quite comprehensive, those for French, Spanish and Dutch are less complete.
- A potential ambiguity with the ring names ‘oxazole’ and ‘thiazole’ has also been resolved. The IUPAC documentation states that it is permissible to omit locants from Hantzsch-Widman names when the locants are consecutive, *i.e.* ‘1,2,3,4-tetrazole’ may be written as ‘tetrazole’, and ‘1,2-oxazirene’ is preferred as ‘oxazirene’. Unfortunately, this conflicts with the traditional interpretations of ‘oxazole’ as meaning ‘1,3-oxazole’ and ‘thiazole’ as ‘1,3-thiazole’. Instead the traditional names ‘isoxazole’ and ‘isothiazole’ denote the ‘1,2-’ forms. This ambiguity, that affected IUPAC-style (but not OpenEye-style) names, has been resolved by preserving the locants, so that the IUPAC names ‘1,2-oxazole’, ‘1,3-oxazole’, ‘1,2-thiazole’ and ‘1,3-thiazole’ are now generated for ‘isoxazole’, ‘oxazole’, ‘isothiazole’ and ‘thiazole’ respectively.

5.8 Lexichem 1.3

- On a benchmark of 250251 compounds in the NCI00 database, `mol2nam` is able to convert 221205 structures (88.39%) to names without BLAH. Of these 221205 names, `nam2mol` is able to convert 183444 (82.93%) back into structures.
- The major announcement of this release is the support for stereochemistry in compound naming. The CIP rules for assigning R/S descriptors to tetrahedral chiral centers, and E/Z descriptors to double bonds are used during name generation.

5.9 Lexichem 1.2

On a benchmark of 250251 compounds in the NCI00 database, `mol2nam` is able to convert 220949 structures (88.29%) to names without BLAH. Of these 220949 names, `nam2mol` is able to convert 182438 (82.57%) back into structures.

5.10 Lexichem 1.2

On a benchmark of 250251 compounds in the NCI00 database, `mol2nam` is able to convert 220949 structures (88.29%) to names without BLAH. Of these 220949 names, `nam2mol` is able to convert 182438 (82.57%) back into structures.

5.11 Lexichem 1.1

- On a benchmark of 250251 compounds in the NCI00 database, `mol2nam` is able to convert 220924 structures (88.28%) to names without BLAH. Of these 220924 names, `nam2mol` is able to convert 177145 (80.18%) back into structures.
- A new `OEIUPACIsLicensed` API has been added so allow applications to check whether *Lexichem*'s parsing and naming functionality can safely be used.

5.11.1 OEParseIUPACName Improvements

The *Lexichem* name parsing routines now handle a small number of structural abbreviations when parsing names. For example, it can now handle names like '3-CF3-5-NO2-benzoic acid'. The usual improvements in name parsing, including more entries for common names in the Lexichem dictionary. Support for names containing multiple explicit hydrogen locants, such as 'pyrimidine-2,4(1H,3H)-dione' and '2,4(1H,3H)-pyrimidinedione'.

5.11.2 OECreatelUPACName Improvements

A serious bug that could cause a core dump when naming thioperoxoic acids has been fixed. The performance of compound naming has been improved. The usual improvements in the names generated (following the IUPAC standards more closely).

5.12 Lexichem 1.0

On a benchmark of 250251 compounds in the NCI00 database, `mol2nam` is able to convert 220922 structures (88.28%) to names without BLAH. Of these 220922 names, `nam2mol` is able to convert 177032 (80.13%) back into structures.

5.12.1 OEParseIUPACName Improvements

In addition to a great many other improvements to the name parsing code, the *Lexichem* parser now contains an internal dictionary allowing the recognition of common non-systematic names, such as ‘ranitidine’ and ‘zantac’.

5.12.2 OECreatelUPACName Improvements

In addition to a great many improvements to the name generation code, the *Lexichem* naming functionality now allows the specification of a naming style, allowing the compound to be named in either a traditional, OpenEye, IUPAC, CAS or systematic naming style.

5.13 Lexichem 1.0.b3

On a benchmark of 250251 compounds in the NCI00 database, `mol2nam` is able to convert 220854 structures (88.25%) to names without BLAH. Of these 220854 names, `nam2mol` is able to convert 144821 (65.57%) back into structures.

5.14 Lexichem 1.0.b2

- On a benchmark of 250251 compounds in the NCI00 database, `mol2nam` is able to convert 208378 structures (83.27%) to names without BLAH. Of these 208378 names, `nam2mol` is able to convert 139100 (66.75%) back into structures.

5.14.1 OEParseIUPACName Improvements

- Support for parsing substituents (name fragments) by generating a wildcard atom in SMILES, for example, ‘methyl’ returns ‘*C’ and ‘hydroxy’ returns ‘*O’.
- Adds support for parenthesized indicated hydrogens, *e.g.* ‘naphthalen-1(2H)-one’.
- Tweaks to support names, such as ‘9,10-anthracene-dicarboxylic acid’, where the hyphen after the stem would cause the parser to expect a list of locants.
- Fixes to multipliers of ‘oxy’ (and related) linkers, so that ‘dineopentyloxybenzene’ is correctly considered two copies of the prefix ‘neopentoxy’.
- Added support for ‘...ic aldehyde’ and ‘...ic acid aldehyde’ as synonyms of ‘...aldehyde’. Added support for both ‘yl’ and ‘oyl’ variants of traditional acid stems, *i.e.* both ‘crotonoyl’ and ‘crotonyl’, and both ‘acryloyl’ and ‘acrylyl’. Adds support for ‘...amido’ as a traditional linker, *e.g.* ‘propanamido’, ‘acrylamido’ and ‘benzamido’.
- Added support for traditional bivalent imides, *e.g.* ‘succinimide’, including N-substituted forms, *e.g.* ‘N-iodosuccinimide’.
- Fixes parsing of salt multipliers, *e.g.* ‘benzene trihydrate’.

- Fixes nitrogenous linker processing, *e.g.* ‘N,N-dimethylsulfamoyl chloride’.
- Adds support for locants alpha, beta and gamma (as written in English), allowing us to handle ‘alpha,alpha,alpha-trichlorotoluene’ and ‘beta-mercaptoethanol’.
- Improvements to handle unspecified locants, *e.g.* ‘chlorophenyl’ and ‘pentachlorophenyl’.
- Support for substitutions on ‘benzoyl’ and ‘benzamido’ prefixes, *e.g.* ‘4-nitrobenzoyl chloride’.
- Handling of names using di, tri etc. incorrectly when bis, tris etc. should have been used, *e.g.* ‘2,4-di(*t*-butylperoxy)hexane’.
- Improvements to ring locant numbers on ‘benzophenone’ and ‘benzidine’, and support for primed locants on ring system parents.
- Adds N and N’ locants to the hydrazine stem, *e.g.* ‘4-(N’,N’-dihydroxyhydrazino)benzoic acid’.
- Treat ‘imidamide’ as a synonym for ‘amidine’, and ‘hydrazinyl’ as a synonym for ‘hydrazino’.
- Adds support for the element ‘columbium’ (more commonly known as ‘niobium’).
- Adds support for the prefixes ‘keto’ (synonym for ‘oxo’), ‘allophanoyl’, ‘hydantoyl’, ‘ureido’, ‘carbamido’, ‘lauryl’, ‘myrsityl’, ‘palmityl’, ‘stearyl’, ‘carbamimidoyl’, ‘sulfinamoyl’, ‘thiocarbamoyl’, ‘carbamothioyl’, ‘carbamoyl’, ‘guanyl’, ‘morpholino’, ‘oxycyano’, ‘sulfinyl’, ‘sulfonyl’ and ‘fulminato’.
- Added support for ‘acetonyl’ and ‘phenacyl’ as vinyl-like prefixes, including their ‘.idene’ and ‘.idyne’ variants.
- Added support for the traditional stems ‘vanillic acid’, ‘isovanillic acid’, ‘syringic acid’, ‘arachidic acid’, ‘benhenic acid’, ‘carboceric acid’, ‘cerinic acid’, ‘ceromelissic acid’, ‘ceroplastic acid’, ‘cerotic acid’, ‘daturic acid’, ‘enanthic acid’, ‘geddic acid’, ‘gheddic acid’, ‘japanic acid’, ‘lacceric acid’, ‘lignoceric acid’, ‘margaric acid’, ‘melissic acid’, ‘montanic acid’, ‘pelargonic acid’, ‘psylic acid’, ‘thapsic acid’, ‘brassylic acid’ and ‘pyruvic acid’.
- Added support for the stems ‘ketene’, ‘thioketene’, ‘vanillin’, ‘isovanillin’, ‘rhodanine’, ‘allophanic acid’, ‘hydantonic acid’, ‘picoline’, ‘borate’ [BH₄-], ‘fulvene’, ‘isobutene’, ‘isoprene’, ‘alloxane’, ‘barbituric acid’, ‘hydantoin’, ‘cytosine’, ‘guanine’, ‘hydroxylamine’ and the common amino acids.
- Adds support for the ring systems ‘benzimidazole’, ‘benzoimidazole’, ‘benzothiophene’, ‘benzoxazole’, ‘benzooxazole’, ‘benzothiazole’, ‘benzotriazole’, ‘benzotrioxazole’, ‘pyrene’, ‘perylene’, ‘as-indacene’, ‘s-indacene’, ‘pyrrolizine’ and ‘quinolizine’.
- Added support for the ring suffix ‘carbonyl chloride’ (and other acid halides).
- Added support for the suffixes ‘thioketone’, ‘sulfide’, ‘nitrite’, ‘azanium’, ‘thial’, ‘diazonium’, ‘arsonic acid’, ‘peroxoic acid’, ‘carboperoxoic acid’, ‘carbothioamide’, ‘carboximidamide’, ‘carboxamidine’, ‘aldehyde oxime’ ‘one oxime’.
- Added support for the linkers ‘mercuri’, ‘carbamimidoyl’, ‘sulfinamoyl’, ‘sulfamoyl’, ‘thiocarbamoyl’, ‘carbamothioyl’, ‘carbonimidoyl’, and ‘thioyl’ (as in ‘ethanethioylbenzene’).
- Added support for the salts ‘hydrobromide’, ‘hydrofluoride’, ‘hydroiodide’, ‘triiodide’, ‘hydrotriiodide’, ‘sulfite’, ‘peroxide’, ‘perchlorate’, ‘perbromate’, ‘periodate’, ‘hydrate’, ‘nitrite’, ‘hypochlorite’, ‘chlorite’, ‘chlorate’, ‘bromate’, ‘iodate’, ‘nitrate’, ‘cyanide’ and ‘cyanate’ (including iso and thio variants).
- Added support for ‘hydrochloric acid’, ‘hydrobromic acid’, ‘hydrofluoric acid’, ‘hydroiodic acid’, ‘hydrotriiodic acid’, and ‘tetric acid’.

5.14.2 OECreatelUPACName Improvements

- Added support for spiro, bicyclo and large simple heterocycle naming, all with hetero replacement nomenclature. No longer elide ring system prefix locants, which fixes the ambiguity with ‘N-tetralinylacetamide’ which should be ‘N-tetralin-1-ylacetamide’.
- Improvements to indicated hydrogen perception (a nitrogen with three ring bonds doesn’t require an indicated hydrogen), improving our naming of ‘indolizine’, ‘pyrrolizine’ and ‘quionlazine’.
- Improvements to naming cycloalkane rings attached via an oxygen linker, *i.e.* ‘cyclopropoxy’ instead of ‘cyclopropyloxy’.
- Improvements in the handling of substituted linkers, ‘carbamoyl’, ‘thiocarbamoyl’, ‘carbamimidoyl’, ‘sulfamoyl’ and ‘sulfanamoyl’.
- Improvements to alkyl chain termination, for example ‘carboxymethyl’ instead of ‘2-hydroxy-2-oxo-ethyl’ and ‘cyanomethyl’ instead of ‘nitridoethyl’.
- Improvements to amide, thioamide and sulfonamide atom typing to allow ‘*C(=O)N=C*’ to be considered an N-substituted amide.
- Added support for more phosphane variants; ‘*=P-*’, ‘*=PH’ and ‘*#P’.
- Add ‘oxamide’ as a contraction of ‘oxalamide’.
- Fixes in multiple suffix processing, *e.g.* ‘cyclohexane-1,4-diamine’.
- Fixes to the prefixes ‘acryloyl’ and ‘proprioloyl’ (and their acid halides) which were previously incorrectly named ‘acrylyl’ and ‘propiolyl’.
- Fixed handling of atom typed metals, fixing ‘trichloromagnesium’.
- Corrected names for bivalent acid halides, *e.g.* ‘malonyl dichloride’.
- Made ‘benzoyl chloride’ the preferred name for ‘benzenecarbonyl chloride’ and likewise for other benzoic acid halides.
- Fix spelling typos in ‘acenaphthene’ and ‘isothiazolidine’.
- Added support for the parents ‘heptalene’, ‘octalene’, ‘hydroxylamine’, ‘hexahydropyrimidine’ (formerly ‘1,2-diazinane’), and ‘hexahydropyridazine’ (formerly ‘1,3-diazinane’).
- Added support for the prefixes ‘acetyl’, ‘aceonylidene’, ‘phenacyl’, ‘phenacylidene’, ‘ureido’, ‘anilino’, ‘hydantoyl’, ‘allophanoyl’, ‘amidino’, ‘acetoxy’, ‘isopropoxy’, ‘isobutoxy’, ‘sec-butoxy’, ‘tert-butoxy’, ‘morpholino’ (formerly ‘morpholin-4-yl’), ‘oxycyano’, ‘sulfinyl’ (*S=O), ‘sulfonyl’ (*S(=O)=O), and ‘methylene’ (formerly ‘methylidene’).
- Added support for the linkers ‘carbonimidoyl’, ‘mercuri’ and ‘benzoyl’ (replacing ‘phenylcarbonyl’).
- Add support for the suffixes ‘amidine’, ‘ohydrazide’, ‘carbohydrazide’, ‘...one oxime’, ‘...al oxime’, ‘aldehyde oxime’ and ‘nitrile oxide’ (note we consider both ‘*C#N=O’ and ‘*C#[N+][O-]’ nitrile oxides).
- Added support for the salts ‘triiodide’, ‘hydrotriiodide’, ‘hydrogen triiodide’, ‘perbromate’, ‘periodate’, ‘sulfide’, ‘sulfite’, ‘peroxide’, ‘iodate’, ‘nitrite’ and ‘hypochlorite’.
- Added support for the molecule ‘fulvene’.

5.15 Lexichem 1.0.b1

On a benchmark of 250,251 compounds in the NCI00 database, `mol2nam` is able to convert 201004 structures (80.32%) to names without BLAH. Of these 201004 names, `nam2mol` is able to convert 111442 (55.44%) back into structures.

5.16 Indices and tables

- *Index*
- *Search Page*

BIBLIOGRAPHY

- [Brecher-1999] J. Brecher, **Name=Struct: A Practical Approach to the Sorry State of Real-Life Chemical Nomenclature**, *Journal of Chemical Information and Computer Science*, Vol. 39, pp. 943–950, 1999.
- [Bünzli-Trepp-2007] Ursula Bünzli-Trepp, **Systematic Nomenclature of Organic, Organometallic and Coordination Chemistry: Chemical-Abstracts Guidelines with IUPAC Recommendations and Many Trivial Names**, EPFL Press, April 2007.
- [Cahn-1966] R.S. Cahn, C.K. Ingold and V. Prelog, **Specification of Molecular Chirality**, *Angew. Chem. Int. Ed. Engl.*, Vol. 5, pp. 385–414, 1966. Errata Vol. 5, p. 511, 1966.
- [Gernot-2006] Gernot A. Eller, **Improving the Quality of Published Chemical Names with Nomenclature Software**, *Molecules*, Vol. 11, pp. 915–928, 2006.
- [Robert-2001] Robert B. Fox and Warren H. Powell, **Nomenclature of Organic Compounds: Principles and Practice**, Oxford University Press publishers, 2001.
- [Garfield-1961] E. Garfield, **Chemico-linguistics: Computer Translation of Chemical Nomenclature**, *Nature*, Vol. 192, pp. 192–194, 1961.
- [Goebels-1991] L. Goebels, A.J. Lawson and J.L. Wisniewski, **AUTONOM: System for Computer Translation of Structural Diagrams into IUPAC-Compatible Names: 2. Nomenclature of Chains and Rings**, *Journal of Chemical Information and Computer Science*, Vol. 31, pp. 216–225, 1991.
- [Hellwinkel-2006] D. Hellwinkel, **Die Systematische Nomenklatur der Organischen Chemie: Eine Gebrauchsanweisung**, Springer publishers, 2006 (German).
- [Hellwinkel-2001] D. Hellwinkel, **Systematic Nomenclature of Organic Chemistry: A Directory to Comprehension and Application of its Basic Principles**, Springer-Verlag publishers, 2001.
- [Nyitrai-1998] József Nyitrai and József Nagy, **Útmutató a szerves vegyületek IUPAC-nevezéktanához**, Magyar Kémikusok Egyesülete, Budapest, 1998 (Hungarian).
- [Labute-1996] Paul Labute, **An Efficient Algorithm for the Determination of Topological RS Chirality**, *Journal of the Chemical Computing Group*, On-line, November 1996.
- [Leigh-2001] G.J. Leigh, H.A. Favre and W.V. Metanomski, **Principes de Nomenclature de la Chemie: Introduction aux recommandations de l'IUPAC**, DeBroeck Université publishers, 2001 (French).
- [Levine-1992] John R. Levine, Ton Mason and Doug Brown, **lex & yacc**, 2nd Edition, UNIX Programming Tools, O'Reilly and Associates publishers, 1992.
- [Mitchell-1948] A.D. Mitchell, **British Chemical Nomenclature**, Edward Arnold & Co. publishers, London, 1948.

- [Peterson-1987] W.R. Peterson, **Formulacion Y Nomenclatura Quimica Organica**, 15th Edition, EDUNSA publishers, Barcelona, **1993**. (Spanish).
- [Polskie-1992] Polskie Towarzystwo Chemiczne, **Nomenklatura Zwiasków Organicznych**, Panstwowe Wydawnictwo Naukowe publishers, Warsaw, **1992**. (Polish).
- [Polskie-1994] Polskie Towarzystwo Chemiczne, **Przewodnik Do Nomenklatury Zwiasków Organicznych**, Narodowy Komitet Miedzynarodowej Unii Chemii Czystej I Stosowanej publishers, Warsaw, **1994**. (Polish).
- [Prelog-1982] V. Prelog and G. Helmchen, **Basic Principles of the CIP-System and Proposals for a Revision**, *Angew. Chem. Int. Ed. Engl.*, Vol. 21, pp. 567–583, **1982**.
- [Sayle-2009] Roger Sayle, **Foreign Language Translation of Chemical Nomenclature by Computer**, *Journal of Chemical Information and Modeling*, Vol. 49, pp. 519–530, **2009** (online: <http://pubs.acs.org/doi/abs/10.1021/ci800243w>)
- [Thurlow-2006] K.J. Thurlow, **Chemical Nomenclature**, Kluwer Academic Publishers, September **1998**.
- [Unicode-2006] The Unicode Consortium, **The Unicode Standard, Version 5.0**, Fifth Edition, Addison-Wesley Professional publishers, October **2006**.
- [Wikman-2004] Susanne Wikman, **Organisk-Kemisk Nomenklatur**, Studentlitteratur Publishers, Lund, **2004**. (Swedish).
- [Williams-1990] Anthony Williams and Andrey Yerin, **The Need for Systematic Naming Software Tools for Exchange of Chemical Information**, *Molecules*, Vol. 4, pp. 255–263, **1999**.
- [Wisniewski-1990] J.L. Wisniewski, **AUTONOM: System for Computer Translation of Structural Diagrams into IUPAC-Compatible Names: 1. General Design**, *Journal of Chemical Information and Computer Science*, Vol. 30, pp. 324–332, **1990**.

INDEX

E

Example Code

mol2nam_example.cpp, 7

nam2mol_example.cpp, 9

M

mol2nam_example.cpp

Example Code, 7

N

nam2mol_example.cpp

Example Code, 9

O

OEIUPAC::OECapitalizeName, 16

OEIUPAC::OECreateIUPACName, 17

OEIUPAC::OEFromChinese, 17

OEIUPAC::OEFromDanish, 17

OEIUPAC::OEFromDutch, 17

OEIUPAC::OEFromFrench, 18

OEIUPAC::OEFromGerman, 18

OEIUPAC::OEFromGreek, 18

OEIUPAC::OEFromHungarian, 18

OEIUPAC::OEFromIrish, 18

OEIUPAC::OEFromItalian, 18

OEIUPAC::OEFromJapanese, 18

OEIUPAC::OEFromKOI8R, 19

OEIUPAC::OEFromLanguage, 19

OEIUPAC::OEFromPolish, 19

OEIUPAC::OEFromRomanian, 19

OEIUPAC::OEFromRussian, 19

OEIUPAC::OEFromSlovak, 19

OEIUPAC::OEFromSpanish, 19

OEIUPAC::OEFromSwedish, 20

OEIUPAC::OEFromUTF8, 20

OEIUPAC::OEFromWelsh, 20

OEIUPAC::OEGetCIPStereo, 20

OEIUPAC::OEIUPAC, 11

OEIUPAC::OEIUPAC::OE_LANG_AMERICAN, 12

OEIUPAC::OEIUPAC::OE_LANG_BRITISH, 12

OEIUPAC::OEIUPAC::OE_LANG_CHINESE, 12

OEIUPAC::OEIUPAC::OE_LANG_DANISH, 12

OEIUPAC::OEIUPAC::OE_LANG_DUTCH, 13

OEIUPAC::OEIUPAC::OE_LANG_ENGLISH, 12

OEIUPAC::OEIUPAC::OE_LANG_FRENCH, 13

OEIUPAC::OEIUPAC::OE_LANG_GERMAN, 13

OEIUPAC::OEIUPAC::OE_LANG_GREEK, 13

OEIUPAC::OEIUPAC::OE_LANG_HUNGARIAN, 13

OEIUPAC::OEIUPAC::OE_LANG_INTERNATIONAL,

12

OEIUPAC::OEIUPAC::OE_LANG_IRISH, 13

OEIUPAC::OEIUPAC::OE_LANG_ITALIAN, 13

OEIUPAC::OEIUPAC::OE_LANG_JAPANESE, 13

OEIUPAC::OEIUPAC::OE_LANG_POLISH, 13

OEIUPAC::OEIUPAC::OE_LANG_PORTUGUESE, 13

OEIUPAC::OEIUPAC::OE_LANG_ROMANIAN, 14

OEIUPAC::OEIUPAC::OE_LANG_RUSSIAN, 14

OEIUPAC::OEIUPAC::OE_LANG_SLOVAK, 14

OEIUPAC::OEIUPAC::OE_LANG_SPANISH, 14

OEIUPAC::OEIUPAC::OE_LANG_SWEDISH, 14

OEIUPAC::OEIUPAC::OE_LANG_WELSH, 14

OEIUPAC::OEIUPAC::OENamStyleACDName, 12

OEIUPAC::OEIUPAC::OENamStyleAutoNom, 12

OEIUPAC::OEIUPAC::OENamStyleCAS, 11

OEIUPAC::OEIUPAC::OENamStyleCASIndex, 11

OEIUPAC::OEIUPAC::OENamStyleIUPAC, 11

OEIUPAC::OEIUPAC::OENamStyleIUPAC79, 12

OEIUPAC::OEIUPAC::OENamStyleIUPAC93, 12

OEIUPAC::OEIUPAC::OENamStyleOpenEye, 11

OEIUPAC::OEIUPAC::OENamStyleSystematic, 11

OEIUPAC::OEIUPAC::OENamStyleTraditional, 11

OEIUPAC::OEIUPACGetArch, 20

OEIUPAC::OEIUPACGetLicensee, 20

OEIUPAC::OEIUPACGetPlatform, 21

OEIUPAC::OEIUPACGetRelease, 21

OEIUPAC::OEIUPACGetSite, 21

OEIUPAC::OEIUPACGetVersion, 21

OEIUPAC::OEIUPACIsLicensed, 21

OEIUPAC::OELanguage, 14

OEIUPAC::OELanguage::AMERICAN, 16

OEIUPAC::OELanguage::BRITISH, 16

OEIUPAC::OELanguage::CHINESE, 16

OEIUPAC::OELanguage::DANISH, 16

OEIUPAC::OELanguage::DUTCH, 16
OEIUPAC::OELanguage::ENGLISH, 16
OEIUPAC::OELanguage::FRENCH, 16
OEIUPAC::OELanguage::GERMAN, 16
OEIUPAC::OELanguage::GREEK, 16
OEIUPAC::OELanguage::HUNGARIAN, 16
OEIUPAC::OELanguage::INTERNATIONAL, 16
OEIUPAC::OELanguage::IRISH, 16
OEIUPAC::OELanguage::ITALIAN, 16
OEIUPAC::OELanguage::JAPANESE, 16
OEIUPAC::OELanguage::POLISH, 16
OEIUPAC::OELanguage::PORTUGUESE, 16
OEIUPAC::OELanguage::ROMANIAN, 16
OEIUPAC::OELanguage::RUSSIAN, 16
OEIUPAC::OELanguage::SLOVAK, 16
OEIUPAC::OELanguage::SPANISH, 16
OEIUPAC::OELanguage::SWEDISH, 16
OEIUPAC::OELanguage::WELSH, 16
OEIUPAC::OELowerCaseName, 21
OEIUPAC::OENAMELocant, 22
OEIUPAC::OEParseIUPACName, 22
OEIUPAC::OEReorderIndexName, 22
OEIUPAC::OESetCIPStereo, 22
OEIUPAC::OEToASCII, 22
OEIUPAC::OEToBritish, 23
OEIUPAC::OEToChinese, 23
OEIUPAC::OEToDanish, 23
OEIUPAC::OEToDutch, 23
OEIUPAC::OEToEUCJP, 23
OEIUPAC::OEToFrench, 23
OEIUPAC::OEToGerman, 23
OEIUPAC::OEToGreek, 24
OEIUPAC::OEToHTML, 24
OEIUPAC::OEToHungarian, 24
OEIUPAC::OEToIrish, 24
OEIUPAC::OEToItalian, 24
OEIUPAC::OEToJapanese, 24
OEIUPAC::OEToLanguage, 24
OEIUPAC::OEToPolish, 25
OEIUPAC::OEToRomanian, 25
OEIUPAC::OEToRussian, 25
OEIUPAC::OEToSJIS, 25
OEIUPAC::OEToSlovak, 25
OEIUPAC::OEToSpanish, 25
OEIUPAC::OEToSwedish, 25
OEIUPAC::OEToUTF8, 26
OEIUPAC::OEToWelsh, 26