

---

# Ogham 2D Chemical Structure Layout and Rendering

*version 1.7*

OpenEye Scientific Software, Inc.

March 19, 2009

9 Bisbee Ct, Suite D  
Santa Fe, NM 87508  
[www.eyesopen.com](http://www.eyesopen.com)  
[support@eyesopen.com](mailto:support@eyesopen.com)

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 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. Alpha is a trademark of Digital Equipment Corporation. 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 Schrödinger, Inc. Schrödinger, Inc may be a wholly owned subsidiary of the Columbia University, New York.

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.

“The forefront of chemoinformatics” is a trademark of Daylight Chemical Information Systems, Inc.

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

# CONTENTS

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Depict</b>	<b>2</b>
2.1	Command Line Interface . . . . .	2
<b>3</b>	<b>Mol2GIF</b>	<b>3</b>
3.1	Command Line Interface . . . . .	3
3.2	Command Line Options . . . . .	3
3.3	Controlling Image Size . . . . .	4
<b>4</b>	<b>Mol2Img</b>	<b>6</b>
4.1	Command Line Interface . . . . .	6
4.2	Command Line Options . . . . .	7
<b>5</b>	<b>Mol2PS</b>	<b>8</b>
5.1	Command Line Interface . . . . .	8
5.2	Command Line Options . . . . .	8
<b>6</b>	<b>OEDepict Classes and Methods</b>	<b>9</b>
6.1	OE8BitImage . . . . .	9
6.2	OEDepictBase . . . . .	10
6.3	OEDepictColor . . . . .	12
6.4	OEDepictView . . . . .	12
6.5	OEPSImage . . . . .	16
6.6	OESVGImage . . . . .	17
<b>7</b>	<b>OEDepict Functions</b>	<b>18</b>
7.1	OEAddDepictionHydrogens . . . . .	18
7.2	OEDepictCoordinates . . . . .	18
7.3	OEDepictFixedCoordinates . . . . .	19
7.4	OEDepictIsLicensed . . . . .	19
7.5	OEHasDepictionHydrogens . . . . .	20
7.6	OEWriteGIF . . . . .	20
7.7	OEWriteBMP . . . . .	21
7.8	OEWriteEPS . . . . .	21
7.9	OEWritePPM . . . . .	21
7.10	OEWriteRGB . . . . .	21

<b>A</b>	<b>depict.cpp Source Code</b>	<b>22</b>
<b>B</b>	<b>Release Notes</b>	<b>23</b>
B.1	Ogham 1.7.0 . . . . .	23
B.2	Ogham 1.6.1, June 2008 . . . . .	23
B.3	Ogham v1.6, February 2008 . . . . .	24
B.4	Ogham v1.5, June 2007 . . . . .	24
B.5	Ogham v1.4, September 2006 . . . . .	25
B.6	Ogham v1.3, February 2006 . . . . .	25
B.7	Ogham v1.2, November 2005 . . . . .	25
B.8	Ogham v1.1, August 2005 . . . . .	26
B.9	Ogham v1.0, June 2005 . . . . .	26
<b>C</b>	<b>Bibliography</b>	<b>27</b>

# Introduction

The OpenEye ogham package includes functionality for generating and rendering 2D co-ordinates of chemical structures.

OpenEye's depiction algorithm assigns 2D co-ordinates to a given molbase, which need not be a single connected structure, using both pattern matching of ring templates, and algorithmic rules for laying out simple ring systems, ring attachments and acyclic chains and functional groups. The depiction algorithm attempts to resolve any clashes in this initial "rule-based" structure using a variety of heuristics.

OpenEye's rendering functionality can display or generate graphical file formats from either the co-ordinates calculated above, or from user provided co-ordinates. Numerous options are available to control the display of atom and bond color, aromaticity, dative bonds, super atoms, etc., in the final image.

# Depict

The `depict` program may be used to assign suitable 2D co-ordinates to a connection table, a process commonly referred to a “depiction” or Structure Diagram Generation (SDG). Input connection tables are read from a variety of file formats, including SMILES, SLN, SD, .mol2 and PDB files, depiction co-ordinates are assigned and the results written to an output molecule file.

## 2.1 Command Line Interface

A description of the command line interface can be obtained by executing `depict` with no arguments.

```
prompt> depict
```

will generate the following output:

```
Depict v1.7 Chemical Structure Layout  
OpenEye Scientific Software, February 2009  
  
usage: depict <molfile> [<outfile>]
```

Command line options are distinguished from filenames by having a ‘-’ prefix. Options can appear anywhere on the command line, *i.e.* before, after or in between filenames. When incompatible options are specified, the last one given on the command line takes effect.

The first filename given on the command line is taken to be the input molecule file, and the optional second filename is treated as the output molecule file. A minus character may be used in place of the input filename to specify that the input is to be read from standard input, `stdin`, and in place of the output filename to specify that the output is to be written to standard output, `stdout`. If only one filename is specified on the command line, the output is written to `stdout` by default.

# Mol2GIF

OpenEye Scientific Software's `mol2gif` application converts molecular structures into bitmap images. If the input molecule file contains 2D co-ordinates (such as an MDL SD file with the "2D" property set in its header), these co-ordinates are used. Otherwise, the `oedepict` library is used to assign suitable 2D co-ordinates.

## 3.1 Command Line Interface

A description of the command line interface can be obtained by executing `mol2gif` with no arguments.

```
prompt> mol2gif
```

will generate the following output:

```
Mol2GIF v1.7 Chemical Structure Renderer  
OpenEye Scientific Software, February 2009  
  
usage: mol2gif [options] <molfile> [<outfile>]
```

Command line options are distinguished from filenames by having a '-' prefix. Options can appear anywhere on the command line, *i.e.* before, after or in between filenames. When incompatible options are specified, the last one given on the command line takes effect.

The first filename given on the command line is taken to be the input molecule file, and the optional second filename is treated as the output image file. A minus character may be used in place of the input filename to specify that the input is to be read from standard input, `stdin`, and in place of the output filename to specify that the output is to be written to standard output, `stdout`. If only one filename is specified on the command line, the output is written to `stdout` by default.

## 3.2 Command Line Options

**-width** Specify the width of the output image in pixels.

**-height** Specify the height of the output image in pixels.

**-gif** Specify CompuServe GIF (.gif) file format output. This is the default.

- bmp** Specify Microsoft BitMap (.bmp) file format output.
- eps** Specify PostScript bitmap (.ps) file format output.
- cob** Specify color-on-black output, instead of the default black-on-white.
- bow** Specify black-on-white rendering. This is the default.
- aromcirc** Display circles in aromatic rings rather than a Kekulé form of alternating single and double bonds.
- noaromcirc** Display a Kekulé form of alternating single and double bonds for aromatic rings. This is the default.
- notitle** Do not display the molecule's title in the image.
- titletop** Display the molecule's title at the top of the depiction.
- titlebot** Display the molecule's title at the bottom of the depiction.
- verbose** Display a CH3 label for terminal methyl groups.
- terse** Do not display labels for terminal methyl groups.
- showh** Display explicit hydrogens.
- noshowh** Suppress explicit hydrogens.
- super** Perform super atom contraction of common functional groups.
- nosuper** Do not perform super atom contraction.
- trans** For GIF image output, mark the background as transparent.
- notrans** Do not mark the background color as transparent in GIF output.
- dative** Display nitro, nitroso and N-oxide groups as an arrow indicating the dative bonding, rather than as a hypervalent or charge separated group.
- nodative** Display nitros, nitrosos and N-oxides as represented in the input structure.
- noauto** Don't automatically magnify molecules to fill drawing area. By default, molecules are magnified or reduced so that they just fit the "canvas" specified by the width and height options.

### 3.3 Controlling Image Size

This section describes how the `mol2gif` program determines a default size for a bitmap image, and how to adjust either the size of the image (using `-width` and `-height`) or its scale (using `-scale` and `-noauto`).

By default, `mol2gif` determines the size of the final image using the input (or calculated) 2D co-ordinates, making each unit in the input co-ordinates equivalent to 25 pixels in the final image. Adding suitable borders then determines the default image size. OEChem's depiction algorithm attempts to make each bond length 1.0, so typically each bond will end up being 25 pixels long.

The entire image can be scaled/resized by using the `-scale` command line option. The default scale (25 pixels per unit) is defined to be 1.0, and the bitmap may be zoomed or shrunk by specifying a floating point value. For example, the option `-scale 0.75` should generate an image a three quarters of the default size, and `-scale 1.5` should generate an image one and a half times the default.

If both the `-wide` and `-high` command line options are used to specify a width and height in pixels, respectively (and the user hasn't specified `-noauto`), the molecule is scaled to maximally fit the given image dimensions.

Otherwise, if the user specifies just the width, or just the height, or both but in conjunction with `-noauto`, these values not only set the final image dimensions, but also shrink the scale so that the molecule fits that dimension. Note, that these values are not used to zoom a molecule; specifying a width and/or height larger than that required by the current scale, causes the molecule to float in the middle of the image of the requested size.

# Mol2Img

The `mol2img` application is an enhanced version of the `mol2gif` program described above, designed to generate 2D image files from connection tables. The difference lies in the fact that `mol2gif` is intended as a pedagogical teaching example of how to use the OpenEye `oedepict` library with its full source code distributed in the `examples/ogham` subdirectory of the Ogham distribution. The `mol2img` program is designed to be a more advanced command line utility, for example automatically selecting the output file format of the image file from its file extension, and supporting additional file formats including PostScript bitmaps, Microsoft BMP, Portable PixMap (PPM) and SGI's RGB file format.

## 4.1 Command Line Interface

A description of the command line interface can be obtained by executing `mol2img` with no arguments.

```
prompt> mol2img
```

will generate the following output:

```
Mol2Img v1.7 Chemical Structure Renderer
OpenEye Scientific Software, February 2009

No argument specified on the command line
Required parameters:
  -in : Input filename
  -out : Output filename
For more help type:
mol2img --help
```

Command line options are distinguished from filenames by having a '-' prefix. Options can appear anywhere on the command line, *i.e.* before, after or in between filenames. When incompatible options are specified, the last one given on the command line takes effect.

The first filename given on the command line is taken to be the input molecule file, and the optional second filename is treated as the output image file. A minus character may be used in place of the input filename to specify that the input is to be read from standard input, `stdin`, and in place of the output filename to specify that the output is to be written to standard output, `stdout`. If only one filename is specified on the command line, the output is written to `stdout` by default.

## 4.2 Command Line Options

- aromcirc** Display circles in aromatic rings rather than a Kekulé form of alternating single and double bonds.
- border** Draw a border around the outside of the image.
- cob** Specify color-on-black output, instead of the default black-on-white.
- dative** Display nitro, nitroso and N-oxide groups as an arrow indicating the dative bonding, rather than as a hypervalent or charge separated group.
- height** Specify the height of the output image in pixels.
- showh** Display explicit hydrogens.
- super** Perform super atom contraction of common functional groups.
- terse** Do not display labels for terminal methyl groups.
- width** Specify the width of the output image in pixels.

# Mol2PS

The `mol2ps` application is a utility for converting a small database of compounds into a Vector postscript file suitable for laser printing or including in publications.

## 5.1 Command Line Interface

A description of the command line interface can be obtained by executing `mol2ps` with no arguments.

```
prompt> mol2ps
```

will generate the following output:

```
Mol2PS v1.7 Chemical Structure Renderer  
OpenEye Scientific Software, February 2009  
  
Usage: mol2ps [-ncol N] [-nrow N] [options] <infile> <out.ps>
```

Command line options are distinguished from filenames by having a '-' prefix. Options can appear anywhere on the command line, *i.e.* before, after or in between filenames. When incompatible options are specified, the last one given on the command line takes effect.

The first filename given on the command line is taken to be the input molecule file, and the optional second filename is treated as the output image file. A minus character may be used in place of the input filename to specify that the input is to be read from standard input, `stdin`, and in place of the output filename to specify that the output is to be written to standard output, `stdout`.

## 5.2 Command Line Options

- ncol** Specify the number of columns of molecules to be displayed per page. The default value is currently two.
- nrow** Specify the number of rows of molecules to be displayed per page. The default value is currently three.
- autofit** Automatically magnify small molecules to fill the display area. By default, small molecules have a default bond length such that most depictions of medicinal chemistry size molecules appear at the same relative magnification in the page. With this option, compounds such as cyclopropane are magnified to fill the available canvas, rather than appear proportional to benzene (for example).

---

# OEDepict Classes and Methods

The “rendering” or “drawing” components of the OEDepict library allow molecules with 2D co-ordinates to be displayed on the screen or written to a number of molecular graphics file formats. This process is conceptually divided into two steps; the first that determines the scale of the depiction, the colors of atoms and bonds, the way in which aromaticity and hydrogens are represented, etc... and the second that converts the graphics primitives generated by the first step into either an in-memory bitmap, a sequence of calls to a graphical drawing library or a sequence of PostScript commands to an output file.

In the OEDepict library, the process of determining how to display a molecule is the task of the OEDepictView class. Once the molecule to display and the various display options have been set the OEDepictView class’s RenderImage method is used to actually output the molecule to the specified OEDepictBase. The OEDepict::OEDepictBase abstract base class defines an interface to (or abstraction of) a graphical rendering API. By deriving classes from OEDepictBase and providing different implementations, the same rendering code in OEDepictView can be used to drive different output devices and/or file formats.

As an example reference implementation of an OEDepictBase, the OEDepict library provides a 8 bit-per-pixel in-memory frame-buffer implementation called OE8BitImage. This can be used to display the molecule into a bitmap. A number of additional functions are also provided to allow the OE8BitImage to be written to a file (or other OEPlatform::oeostream) in several popular file formats including CompuServe GIF, Microsoft Bitmap (BMP), Portable Pixmap (PPM), PostScript Bitmap, and SGI’s RGB file format.

## 6.1 OE8BitImage

```
class OE8BitImage : public OEDepictBase
```

The OE8BitImage class is an implementation of the abstract OEDepictBase base class (interface) that implements the drawing functionality by writing into an 8 bit-per-pixel in-memory bitmap.

### 6.1.1 Constructors

```
OE8BitImage()
```

Default constructor. By default, the width and height of an OE8BitImage are 200 pixels wide by 200 pixels high. The current width and height can be determined using the OEDepictBase::GetXSize and OEDepictBase::GetYSize methods. If required, the dimensions of an OE8BitImage may be adjusted using the OEDepictBase::Resize method.

```
OE8BitImage(int width, int height)
```

Construct with a specified width and height. The current width and height can be determined using the `OEDepictBase::GetXSize` and `OEDepictBase::GetYSize` methods. If required, the dimensions of an `OE8BitImage` may be adjusted using the `OEDepictBase::Resize` method.

## 6.2 OEDepictBase

The `Clip*` methods are just bounds checking versions of the `Draw*` methods. Only use the `Draw*` methods if it is known that the bounds are being obeyed. The methods with the `*F` suffix take floating point arguments instead of integers. This only matters for formats that can take floating point measurements (eg: postscript), otherwise the default is to round to an integer and call the integer equivalent.

```
class OEDepictBase
```

The `OEDepictBase` abstract base class defines a device-independent interface for common 2D drawing operations, such as drawing points, lines, rectangles and circles.

### 6.2.1 Clear

```
void Clear(int col)
```

Clear the current drawing area to the specified color.

The color argument, `col`, should be a color handle returned by the `OEDepictBase::Color` method for this class.

### 6.2.2 ClipCenterString

```
virtual void ClipCenterString(int x, int y, const char *str, int c, int dx)
```

Place a string `str` of length `dx` at the co-ordinates `x1,y1` in the color `c`.

### 6.2.3 ClipCenterStringF

```
virtual void ClipCenterStringF(float x, float y, const char *str,
                               int c, float dx)
```

Same as `ClipCenterString`, but takes floating point values.

### 6.2.4 ClipLine

```
void ClipLine(int x1, int y1, int x2, int y2, int c)
```

Draw a one pixel wide line from the co-ordinates `x1,y1` to `x2,y2` with color `c`.

The co-ordinates for this function need not be within the current drawing area, in which case this operation will clip the primitive to the current drawing area.

The color argument, *c*, should be a color handle returned by the `OEDepictBase::Color` method for this class.

### 6.2.5 ClipPoint

```
void ClipPoint(int x, int y, int c)
```

Plot a one pixel point at the co-ordinates *x,y* with color *c*.

The co-ordinates for this function need not be within the current drawing area, in which case this operation will clip the primitive to the current drawing area.

The color argument, *c*, should be a color handle returned by the `OEDepictBase::Color` method for this class.

### 6.2.6 Color

```
int Color(unsigned char r, unsigned char g, unsigned char b)
```

Retrieve (or assign) a suitable color handle (index) for a specified color. The required color is specified by the *r*, *g* and *b* arguments which specify the 8-bit red, green and blue components of the color. The returned value provides the “color handle” to be passed to the other drawing functions. The encoding of colour is implementation specific, for example, an index into a color palette for some drivers.

### 6.2.7 DrawLine

```
void DrawLine(int x1, int y1, int x2, int y2, int c)
```

Draw a one pixel wide line from the co-ordinates *x1,y1* to *x2,y2* with color *c*.

The co-ordinates for this function must all be within the current drawing area.

The color argument, *c*, should be a color handle returned by the `OEDepictBase::Color` method for this class.

### 6.2.8 DrawPoint

```
void DrawPoint(int x, int y, int c)
```

### 6.2.9 GetXSize

```
int GetXSize() const
```

Get the current width of the `OEDepictBase` in pixels. The dimensions of the `OEDepictBase` drawing area may be specified either when the class is constructed or by using the `OEDepictBase::Resize` method.

### 6.2.10 GetYSize

```
int GetYSize() const
```

Get the current height of the `OEDepictBase` in pixels. The dimensions of the `OEDepictBase` drawing area may be specified either when the class is constructed or by using the `OEDepictBase::Resize` method.

### 6.2.11 Resize

```
bool Resize(int x, int y)
```

Attempt resize the current drawing area to the specified dimensions. The `x` and `y` arguments denote the required width and height in pixels respectively. This function returns true, if the operation was successful, and false if it failed.

Some `OEDepictBase` implementations prohibit the resizing once drawing has started. Some implementations attempt to preserve the contents of the current drawing area, whilst others implicitly clear their contents (using `OEDepictBase::Clear`) upon resizing.

## 6.3 OEDepictColor

```
struct OEDepictColor
{
    unsigned char r, g, b;
}
```

This structure is used by the `OEDepict` library to store a Red-Green-Blue (RGB) color triple. Currently, the only place this is externally visible is the `col` array of an `OE8BitImage`. The `r`, `g` and `b` fields hold the 8-bit red, green and blue components of a 24-bit color.

## 6.4 OEDepictView

```
class OEDepictView
```

The `OEDepictView` class provides all of the knowledge (and state) for drawing a 2D molecular depiction.

### 6.4.1 Constructors

```
OEDepictView()
```

Default constructor. By default, the width and height of an `OEDepictView` are 200 pixels wide by 200 pixels high. The current width and height can be determined using the `OEDepictView::XRange` and `OEDepictView::YRange` methods. If required, the dimensions of an `OEDepictView` may be adjusted using either the `OEDepictView::AdjustView` or `OEDepictView::DetermineView` methods.

```
OEDepictView(int width, int height)
```

Construct with a specified width and height. The current width and height can be determined using the `OEDepictView::XRange` and `OEDepictView::YRange` methods. If required, the dimensions of an `OEDepictView` may be adjusted using either the `OEDepictView::AdjustView` or `OEDepictView::DetermineView` methods.

```
OEDepictView(const OEDepictView &copy)
```

Copy constructor.

## 6.4.2 AdjustView

```
void AdjustView(int wide, int high)
```

Set the dimensions of the view of the current molecule to the specified dimensions. The `wide` argument specifies the width of the view in pixels, and the `high` argument specifies the height of the view in pixels. This method must be called after the molecule to be displayed has been set by the `OEDepictView::SetMolecule` method.

This function sets the `xscale`, `yscale`, `xoffset` and `yoffset` properties of the `OEDepictView`, as can be retrieved by the methods `OEDepictView::XScale`, `OEDepictView::YScale`, `OEDepictView::XOffset` and `OEDepictView::YOffset`.

## 6.4.3 AStyle

```
OEAtomStyle *AStyle(unsigned int i)
```

Return a pointer to the “atom style” structure associated with a particular atom of the molecule currently associated with the `OEDepictView`. The `i` argument is the atom index of the atom, as returned by `OEAtomBase::GetIdx`. If no molecule is associated with the `OEDepictBase`, or the atom index `i` is out of range for the current molecule, this method returns a NULL pointer.

A molecule may be associated with the `OEDepictView` using the method `OEDepictView::SetMolecule`.

## 6.4.4 BStyle

```
OEBondStyle *BStyle(unsigned int i)
```

Return a pointer to the “bond style” structure associated with a particular bond of the molecule currently associated with the `OEDepictView`. The `i` argument is the bond index of the bond, as returned by `OEBondBase::GetIdx`. If no molecule is associated with the `OEDepictBase`, or the bond index `i` is out of range for the current molecule, this method returns a NULL pointer.

A molecule may be associated with the `OEDepictView` using the method `OEDepictView::SetMolecule`.

## 6.4.5 DetermineView

```
void DetermineView(int wide, int high)
```

The `DetermineView` method can be used to set the drawing area of the `OEDepictView` to the size preferred by the currently associated molecule. This method must be called after the molecule to be displayed has been set by the `OEDepictView::SetMolecule` method.

If both the `wide` and `high` arguments are both zero, the width and height of the depiction, as retrieved using the `OEDepictView::XRange` and `OEDepictView::YRange` methods, as set to snugly fit around the depiction at the current ‘pixel scale’. The ‘pixel scale’ corresponds to the number of pixels represented to a unit length in the molecule’s co-ordinate system. The default pixel scale is 28 pixels, hence a bond of length 1.0 will be displayed as a line 28 pixels long. The pixel scale can get modified and retrieved using the `OEDepictView::SetPixScale` and `OEDepictView::GetPixScale` methods, respectively.

If either the `wide` or the `high` arguments are non-zero, they specify the width and height bounds to be used. If these values are larger than the size that would be chosen for the molecule, the molecule is displayed at its default

scale centered within a larger box. Otherwise, the molecule is shrunk to fit in the specified size. This functionality is useful in spreadsheets and WWW pages where a depiction may have to fit within a particular column width but may be of any height.

This function also sets the `xscale`, `yscale`, `xoffset` and `yoffset` properties of the `OEDepictView`, as can be retrieved by the methods `OEDepictView::XScale`, `OEDepictView::YScale`, `OEDepictView::XOffset` and `OEDepictView::YOffset`.

### 6.4.6 GetBackColor

```
void GetBackColor(unsigned char *r,  
                 unsigned char *g,  
                 unsigned char *b)
```

Retrieve the current background color of the `OEDepictView`. Upon return, the 8-bit red, green and blue components of the background color are stored in the variables pointed to by `r`, `g` and `b` arguments respectively.

The background color may be specified using the `OEDepictView::SetBackColor` method.

### 6.4.7 GetFontSize

```
int GetFontSize()
```

### 6.4.8 GetForeColor

```
void GetForeColor(unsigned int *r,  
                 unsigned int *g,  
                 unsigned int *b)
```

Retrieve the current foreground color of the `OEDepictView`. Upon return, the 8-bit red, green and blue components of the foreground color are stored in the variables pointed to by `r`, `g` and `b` arguments respectively.

The foreground color may be specified using the `OEDepictView::SetForeColor` method.

### 6.4.9 GetMolecule

```
const OEChem::OEMolBase *GetMolecule()
```

Return a pointer to the molecule currently associated with this `OEDepictView` or a `NULL` pointer if no molecule has been set. The molecule associated with a `OEDepictView` may be set with the method `OEDepictView::SetMolecule`. By default, no molecule is associated a `OEDepictView`.

### 6.4.10 GetTitleSize

```
int GetTitleSize()
```

### 6.4.11 GetTitleTop

```
bool GetTitleTop()
```

### 6.4.12 RenderImage

```
void RenderImage(OEDepictBase *img, bool clear=true,  
                int startx = 0, int starty = 0)
```

Display the current molecule using the given OEDepictBase, img. The clear argument controls whether the drawing area should be cleared before displaying the molecule (by default true), and the startx and starty arguments allow the image to display at the specified offset withing the specified OEDepictBase.

### 6.4.13 SetBackColor

```
void SetBackColor(unsigned char r,  
                  unsigned char g,  
                  unsigned char b)
```

### 6.4.14 SetFontSize

```
bool SetFontSize(int size)
```

### 6.4.15 SetForeColor

```
void SetForeColor(unsigned char r,  
                  unsigned char g,  
                  unsigned char b)
```

### 6.4.16 SetMolecule

```
void SetMolecule(const OEChem::OEMolBase *mol)
```

Set the molecule to be displayed. A pointer is kept to the original molecule, but the molecule itself is neither modified nor copied. Great care should be used when modifying the molecule associated with a OEDepictView. After modifying a molecules, for example, by adding or deleting atoms or bonds, or modifying co-ordinates, this SetMolecule function should be called again. A pointer to the current molecule associated with a OEDepictView may be retrieved using the OEDepictView::GetMolecule method.

This function allocates (or reallocates) the 'atom style' and 'bond style' arrays associated with the molecule. These arrays may be retrieved and modified using the OEDepictView::AStyle and OEDepictView::BStyle methods.

Passing a NULL pointer to SetMolecule can be used to reset the association between the OEDepictView and a molecule. It is not necessary to do this explicitly prior to destroying the OEDepictView.

### 6.4.17 SetTitleSize

```
void SetTitleSize(int size)
```

### 6.4.18 SetTitleTop

```
void SetTitleTop(bool top)
```

#### 6.4.19 XOffset

```
int XOffset()
```

#### 6.4.20 XRange

```
int XRange()
```

#### 6.4.21 XScale

```
int XScale()
```

#### 6.4.22 YOffset

```
int YOffset()
```

#### 6.4.23 YRange

```
int YRange()
```

#### 6.4.24 YScale

```
int YScale()
```

### 6.5 OEPSImage

```
class OEPSImage : public OEDepictBase
```

The OEPSImage class is an implementation of the abstract OEDepictBase base class (interface) that implements the drawing functionality by generating Encapsulated PostScript.

#### 6.5.1 Write

```
bool Write(OEPlatform::oeostream &os)  
bool Write(OEPlatform::oeostream &os, bool more)
```

After rendering an image into a OEPSImage instance, use the OEPSImage::Write method to emit the PostScript commands to the specified output stream os.

The second form of the OEPSImage::Write function can be used to efficiently generate multiple page PostScript documents. If the more argument is true, the current page will be written assuming that there are

more pages to follow. The last page of a multiple document should be emitted with a `more` value of `false`, in order to generate the appropriate PostScript trailers.

The single parameter form is equivalent to calling the two parameter form with a second argument of `false`.

## 6.6 OESVGImage

```
class OESVGImage : public OEDepictBase
```

The `OESVGImage` class is an implementation of the abstract `OEDepictBase` base class (interface) that implements the drawing functionality by generating Scalable Vector Graphics (SVG) commands, which are a form of XML.

### 6.6.1 Write

```
bool Write(OEPlatform::oeostream &os, bool header=true)
```

After rendering an image into a `OESVGImage` instance, use the `OESVGImage::Write` method to emit the SVG commands to the specified output stream `os`. The `header` argument can be used to control whether the initial “`<?xml ...?>`” and “`<!DOCTYPE svg ...>`” tags are written.

# OEDepict Functions

This chapter describes the OEDepict C++ API. All of these functions reside in the OEDepict namespace, but operate on OEChem molecules.

## 7.1 OEAddDepictionHydrogens

```
bool OEAddDepictionHydrogens(OEMolBase &mol)
```

This function adds explicit “depiction” hydrogens to the specified OEMolBase. Depiction hydrogens are hydrogens that need to be explicitly drawn in a 2D depiction to faithfully represent tetrahedral stereochemistry. A depiction hydrogen is currently defined as the single hydrogen attached to a non-hydrogen atom that has tetrahedral chirality specified and either has only two non-hydrogen neighbors, or has three non-hydrogen neighbors that are connected via cyclic “ring” bonds. See the definition of OEHasDepictionHydrogens below. For example, the hydrogens on the fusion atoms of decalin are depiction hydrogens.

The OEAddDepictionHydrogens function identifies all implicit depiction hydrogens (using the OEHasDepictionHydrogens function), and converts them into explicit hydrogens using the OEAddExplicitHydrogens function. This function returns true if any depiction hydrogens were converted into explicit atoms.

```
bool OEAddDepictionHydrogens(OEMolBase &mol)
{
    bool result = false;
    OEIter<OEAtomBase> atom;
    for (atom=mol.GetAtoms(); atom; ++atom)
    {
        OEAtomBase *aptr = atom;
        if (OEHasDepictionHydrogens(aptr) &&
            OEAddExplicitHydrogens(mol,aptr))
            result = true;
    }
    return result;
}
```

## 7.2 OEDepictCoordinates

```
bool OEDepictCoordinates(OEMolBase &mol)
```

This function assigns 2D co-ordinates to the given OEMolBase. The co-ordinates of each explicit atom are assigned using OEChem's OEAtomBase::SetCoords method. This function automatically set the dimensionality of the molecule to two, using the OEMolBase::SetDimension.

This function returns true if 2D co-ordinates were successfully assigned to each atom. In the current implementation, this function always returns true.

## 7.3 OEDepictFixedCoordinates

```
bool OEDepictFixedCoordinates(OEChem::OEMolBase &mol,
                              const unsigned char *afixed,
                              const unsigned char *bfixed = 0)
```

This function is an extended for of the OEDepictCoordinates API, that allows the user to specify or “fix” a subset of atoms of the current molecule. The `afixed` input parameter must be a valid pointer to the start of an array at least `mol.GetMaxAtomIdx()` bytes long, and the `bfixed` input parameter, if specified, must be a valid pointer to the start of an array at least `mol.GetMaxBondIdx()` bytes long.

For each atom in the molecule, a non-zero value for `afixed[atm->GetIdx()]` indicates that it's “lay-out” should be fixed, and a zero value indicates that the depiction algorithm should provide co-ordinates. When `bfixed` is not specified, a bond `bnd` is considered fixed when both of its end atoms are fixed, *i.e.* `afixed[bnd->GetBgnIdx()] && afixed[bnd->GetEndIdx()]`. However, there are times when finer control is required, and the `bfixed` array allows the caller to specify which bonds are fixed. This is useful for in applications such as using ring templates, where in molecules such as biphenyl, all the atoms may be given co-ordinates, but the non-ring bond linking them may not have been “fixed” and so OEDepictCoordinates is expected to arrange the two rings relative to each other.

If a bond is fixed, its distance (bond length) is preserved by this function. Likewise, for two fixed bonds that share a common fixed atom, the bond angle between them is preserved. If all of the fixed atoms of a molecule are “connected” by fixed bonds, then the co-ordinates of the fixed atoms should be preserved on output (perhaps with minor differences due to rounding errors). If the fixed substructures of a molecule aren't connected, the orientation of the first is preserved, but the relative layout (lengths and angles) of the others are preserved. Hence, if all the atoms and bonds of a molecule are fixed, this function should return (approximately) the same co-ordinates as the input molecule. If all of the atoms and bonds are unfixed, this function should be equivalent to calling OEDepictCoordinates.

This function returns true if 2D co-ordinates were successfully assigned to every atom. In the current implementation, this function fails if the input molecule is a disconnected structure (*i.e.* multiple parts) or if the molecule's dimension (as returned by the method `OEMolBase::GetDimension`) isn't two. Future releases may relax the single connected component restriction.

## 7.4 OEDepictIsLicensed

```
bool OEDepictIsLicensed(const char *features = 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 OEDepict's functionality.

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

```
if (!OEDepictIsLicensed("python"))
```

```
OEThrow.Warning("OEDepict 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 (OEDepictIsLicensed(0, expdate))
{
    OEThrow.Info("License expires: day: %d month: %d year: %d",
                expdate[0], expdate[1], expdate[2]);
}
```

## 7.5 OEHasDepictionHydrogens

```
bool OEHasDepictionHydrogens(const OEAtomBase *atm)
```

This function determines whether the specified heavy atom has a "depiction" hydrogen. Depiction hydrogens are hydrogens that need to be explicitly drawn in a 2D depiction to faithfully represent tetrahedral stereochemistry. A depiction hydrogen is currently defined as the single hydrogen attached to a non-hydrogen atom that has tetrahedral chirality specified and either has only two non-hydrogen neighbors, or has three non-hydrogen neighbors that are connected via cyclic "ring" bonds. See the implementation of `OEHasDepictionHydrogens` below. For example, the fusion atoms of decalin have depiction hydrogens.

```
bool OEHasDepictionHydrogens(const OEAtomBase *atm)
{
    if (atm->GetAtomicNum() == 1)
        return false;
    if (atm->GetTotalHCount() != 1)
        return false;

    unsigned int degree = atm->GetHvyDegree();
    if (atm->HasStereoSpecified())
    {
        if (degree == 2)
            return true;
        if (degree == 3 && RingBondCount(atm) == 3)
            return true;
    }

    if (degree == 1)
    {
        OEBondBase *bptr = OEGetSoleDoubleBond(atm);
        if (bptr && bptr->HasStereoSpecified(OEBondStereo::CisTrans))
            return true;
    }
    return false;
}
```

## 7.6 OEWriteGIF

```
bool OEWriteGIF(OEPlatform::oeostream &os, const OE8BitImage &img,
                int transparent=-1)
```

Write the pixels of the specified bitmap image `img` to the output stream `os` in GIF file format.

## 7.7 OEWriteBMP

```
bool OEWriteBMP(OEPlatform::oeostream &os, const OE8BitImage &img)
```

Write the pixels of the specified bitmap image `img` to the output stream `os` in Microsoft BMP file format.

## 7.8 OEWriteEPS

```
bool OEWriteEPS(OEPlatform::oeostream &os, const OE8BitImage &img)
```

Write the pixels of the specified bitmap image `img` to the output stream `os` in Encapsulated PostScript file format. Note that this function generates a PostScript file containing a bitmap. To generate a PostScript file containing vector graphics primitives render the molecule into an `OEPSImage` class instead of an `OE8BitImage`.

## 7.9 OEWritePPM

```
bool OEWritePPM(OEPlatform::oeostream &os, const OE8BitImage &img,  
                bool binary=true)
```

Write the pixels of the specified bitmap image `img` to the output stream `os` in Portable PixMap (PPM) file format.

## 7.10 OEWriteRGB

```
bool OEWriteRGB(OEPlatform::oeostream &os, const OE8BitImage &img)
```

rite the pixels of the specified bitmap image `img` to the output stream `os` in Silicon Graphics' RGB file format.

## Depict.cpp Source Code

```
#include "openeye.h"
#include "oechem.h"
#include "oedepict.h"

using namespace OESystem;
using namespace OEChem;
using namespace OEDepict;

int main(int argc, char *argv[])
{
    if (argc != 3) OETHrow.Usage("depict <infile> <outfile>");
    oemolistream ifs(argv[1]);
    if(!ifs) OETHrow.Fatal("Unable to open %s for reading",argv[1]);
    oemolostream ofs(argv[2]);
    if(!ofs) OETHrow.Fatal("Unable to open %s for writing",argv[2]);

    OEGraphMol mol;
    while (ifs >> mol) {
        if (mol.GetDimension() == 3) {
            OEPerceiveChiral(mol);
            OE3DToBondStereo(mol);
            OE3DToAtomStereo(mol);
        }
        OEAddDepictionHydrogens(mol);
        OEDepictCoordinates(mol);
        OEMDLPerceiveBondStereo(mol);
        ofs << mol;
    }
    return 0;
}
```

# Release Notes

## B.1 Ogham 1.7.0

1. Improvements have been made to the co-ordinate generation routines to consider stretching bonds in congested (per-substituted) chains, and to terminal atoms.
2. The generation of linear carbon atoms is now avoided, even in strained systems such as methanedisulfonic acid. Depending upon the presence of heteroatomic neighbors and/or the use of color, such linear carbons were ambiguous, potentially being interpreted as stretched bonds. Instead, the central atom remains crinkled, but the bonds to it are elongated to avoid overlaps.
3. To avoid potential ambiguities with linear carbon atoms, the rendering routines now display a label on linear carbon atoms, typically CH<sub>2</sub>. Similar functionality already existed to display a label in allenic systems.
4. Support for using an anti-aliased bitmapped font for atom labels and titles has been added to the default 8-bit image renderer, `OE8BitImage`. This replaces the vector fonts used previously. Text functions now use a proportionally spaced sans serif font.
5. New `SetAntiAliased` and `GetAntiAliased` methods have been added to the `OE8BitImage` class to control the use of (text) anti-aliasing. Although, anti-aliasing is enabled by default, this can create edge artifacts when generating transparent bitmap images, if the display background color doesn't match the background color used for anti-aliasing. In such cases, `SetAntiAliased(false)` can be used to explicitly turn off antialiasing.
6. The rendering of atom labels on free (disconnected) atoms has been improved, to avoid being displaced by a hydrogen count in the label. Previously, the presence of the hydrogen count in H<sub>2</sub>O would cause picking and highlighting operations to emphasize the H instead of the O.
7. Improvements have been made to the algorithm for choosing which side of a double bond to display the second line. This improves the display of molecules like [18]annulene.
8. Many new macrocycle ring systems have been added to the ring template dictionary, and existing entries have been updated with substitution preferences.

## B.2 Ogham 1.6.1, June 2008

1. Added `OESVGImage` support for Python and Java

2. Added optional parameter to `OESVGImage` whether to write out the appropriate XML header tags.

### B.3 Ogham v1.6, February 2008

1. Added a new `OESVGImage` class to allow depictions to be written out in Scalable Vector Graphics (`.svg`) file format. This provides a convenient vector graphics file format, similar to the existing Encapsulated PostScript writer, but the resulting format is easier to parse by applications.
2. Support for the `.svg` file extension has been added to the Ogham application `mol2img` (and the `mol2img.cpp` example source code).
3. Two new example source files have been added to Ogham toolkit, `mol2png` and `mol2jpg`. These demonstrate how the Ogham toolkit can be used with the open source libraries, `libpng` and `libjpeg` respectively. PNG (or Portable Network Graphics) file format is a lossless compression image file format, whilst JPEG (or Joint Photographic Experts Group) is a lossy compression image file format. The corresponding libraries are installed on most Linux machines, or can be downloaded from the following sites.

<http://www.libpng.org/pub/png/libpng.html>  
<http://www.iwg.org/>

4. Improved depiction support for reactions. Previously, reactions would be treated identically to disconnected mixtures so "A>>B" would be given identical co-ordinates to "A.B". We now add extra spacing to separate the reactants from the products, to prevent the reaction arrows drawn by Ogham's renderer from overlapping any atoms.
5. Several new ring systems have been added to the ring template dictionary, and existing entries have been updated with substitution preferences.
6. Fixed a minor bug in Ogham's renderer where we accidentally permuted the red, green and blue components of the background color when displaying text. This only affected depictions that used background colors other than black or white (or monochromatic grey).

### B.4 Ogham v1.5, June 2007

1. Improved functionality in `OEDepictFixedCoordinates` allows finer grain control over user-specified layout by providing an optional `bfixed` argument in addition to the existing `afixed` array. This allows, for example, the ability for the user to specify how two ring systems connected by a linking bond are laid out, but require the function to orient them relative to one another.
2. Several new ring systems have been added to the ring template dictionary, and existing entries have been updated with substitution preferences, and the new concave attachment annotations.
3. The Adobe PostScript driver in `mol2ps` has been tweaked to emit Encapsulated PostScript (EPSF) only for single page documents, and to use regular PostScript for multiple page output. This avoids issues with Apple's "Preview" application, which assumes that EPSF files never have more than a single page. This change has the added benefit of resulting in smaller PostScript files.
4. A bug was fixed in `OEDepictFixedCoordinates` where if the first fixed atom had degree 1, the orientation of the original depiction would get lost, even though the bond lengths and bond angles were preserved.

## B.5 Ogham v1.4, September 2006

1. Improved clash resolution to ease atomic overlap in congested depictions. The algorithm now tries harder when flipping all pairs of bonds to prevent clashes, and is also more intelligent about bond stretching.
2. The heuristics for degree four centers now prefer to initially keep congested atoms and/or ring atoms in opposition.
3. Improved ring template handling so that concave atoms of degree two are annotated such that substitutions sprout into the smaller sector.
4. Improved handling of ring system substitution preferences, such that ring atom positions with multiple substitutions are now weighted more heavily when selecting the most suitable automorphism/orientation.
5. Numerous new ring systems have been added to the ring template dictionary, and existing entries have been updated with substitution preferences, and the new concave attachment annotations.
6. Fixed a minor bug which due to floating point rounding could call the system's `acos` function with values fractionally less than `-1.0`. This resulted in different behavior on different platforms, such that some Ogham depictions differed between Linux and Solaris.
7. Fixed another minor bug that could affect complex ring systems containing multiple spiro atoms.
8. Numerous refinements to the rendering functionality to better defend against incorrect usage.

## B.6 Ogham v1.3, February 2006

1. This is predominantly a maintenance release to provide a version of the `oedepict` library compatible with OEChem v1.4. There have however been a number of additions and improvements to the internal ring template dictionary. Improved `OECanonicalOrderAtoms` and `OECanonicalOrderBonds` functionality in OEChem v1.4 allows for the generation of “canonical” depictions, that are less affected by input atom ordering.

## B.7 Ogham v1.2, November 2005

1. The definition of a depiction hydrogen has been extended to include hydrogens necessary to represent cis/trans stereochemistry for imines.
2. Unnamed transuranic elements are now displayed as `#n` where `n` is the atomic number.
3. Improvements to the aesthetics of acyclic double bonds between two labeled atoms, such as azo linkers. The double bond is now displayed centered instead of offset to one side.
4. Added support for a new “bowtie” bond-style and a new `bowties` property of the `OEDepictView` class to enable the display of chiral double bonds without specified cis/trans stereochemistry.
5. Fixed a potential crash when drawing aromatic circles to a bitmap, where the atomic coordinates result in a part of the circle being drawn outside the bitmap's bounds.

## B.8 Ogham v1.1, August 2005

1. Improvements to depictions of reactions, including the display of a “reaction arrow” between the reactants and products.
2. Bonds may now be colored independently of the atoms they connect.
3. An improved algorithm for the placement (north, south, east or west) of implicit hydrogens in atomic labels.
4. Carbon atoms in strange valence states (such as radicals) are now displayed with a label (including implicit hydrogen count).
5. Attachment point atoms read from MDL file formats (such as .mol, .rxn and .sd) are now displayed with the attachment index, i.e. “R2”, instead of just “R#”. This matches the behavior of MDL’s ISIS Draw.
6. A new `OEDepictIsLicensed` API point to determine whether the Ogham co-ordinate generation and image rendering functionality is available to an application.
7. Additions to the internal ring template dictionary.

## B.9 Ogham v1.0, June 2005

1. Ability to “freeze” or specify the layout of a substructure of a molecule and request that Ogham layout the rest of the molecule. This use useful for preserving the orientation of the cores of combinatorial libraries or the matching atoms of a substructure search.
2. Add support for specifying substitution preferences via the ring template library.
3. Increased ring template dictionary, including both style and orientation preferences. Additionally, a number of existing ring templates have been enhanced with substitution preferences.
4. Improved crinkling of acyclic chain systems.
5. New clash resolution protocol that allows stretching of acyclic bonds between two ring systems.
6. The bond lengths of bonds to non-depiction hydrogens has now been reduced to one half, to avoid cluttering depictions that include explicit hydrogens.
7. Improvements to the documentation and minor bug fixes.

---

## Bibliography

1. Alex M. Clark, Paul Labute and Martin Santavy, “2D Structure Depiction”, *Journal of Chemical Information and Modeling (JCIM)*, Vol. 46., No. 3, pp. 1107–1123, 2006.
2. Paul G. Dittmar, Joseph Mockus and Kathryn M. Couvreur, “An Algorithmic Computer Graphics Program for Generating Chemical Structure Diagrams”, *Journal of Chemical Information and Computer Sciences (JCICS)*, Vol. 17., No. 3, pp. 186–192, 1977.
3. Patrick C. Fricker, Marcus Gastreich and Matthias Rarey, “Automated Drawing of Structural Molecular Formulas under Constraints”, *Journal of Chemical Information and Computer Sciences (JCICS)*, Vol. 44., No. 3, pp. 1065–1078, 2004.
4. Alan L. Goodson, “Graphical Representation of Chemical Structures in Chemical Abstract Service Publications”, *Journal of Chemical Information and Computer Sciences (JCICS)*, Vol. 20, No. 4, pp. 212–217, 1980.
5. Harold E. Helson, “Structure Diagram Generation”, Chapter 6, pp. 313–398, in “Reviews in Computational Chemistry”, Vol. 13, Kenny B. Lipkowitz and Donald B. Boyd (Editors), Wiley-VCH Press, 1999.
6. Craig A. Shelley, “Heuristic Approach for Displaying Chemical Structures”, *Journal of Chemical Information and Computer Sciences (JCICS)*, Vol. 23, No. 2, pp. 61–65, 1983.
7. Andrew C. Wallace, Roman A. Laskowski and Janet M. Thornton, “LIGPLOT: A Program to Generate Schematic Diagrams of Protein-Ligand Interactions”, *Protein Engineering*, Vol. 8, No. 2, pp. 127–134, 1995.
8. David Weininger, “SMILES 3: Depict - Graphical Depiction of Chemical Structures”, *Journal of Chemical Information and Computer Sciences (JCICS)*, Vol. 30, No. 3, pp. 237–243, 1990.