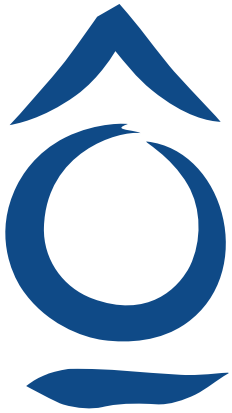

OpenEye
SCIENTIFIC

# VIDA Scripting Manual

***Release 4.4.0.4***

**OpenEye Scientific Software, Inc.**

February 15, 2018

CONTENTS

CHAPTER

ONE

# GENERAL FUNCTIONS

The functions defined in this section provide a wide variety of functionality that is generally applicable to the application as a whole.

## 1.1 About

```
void About()
```

Displays information about VIDA.

## 1.2 AppAddCallback

```
unsigned int AppAddCallback(changeName, callbackFunction)
```

Adds a Python callback function to be called when certain changes occur within VIDA. The changeName parameter is a string, and can have the following values:

- "ActiveChanged"
- "DisplayChanged"
- "LockedChanged"
- "MarkedChanged"
- "ObjectsAdded"
- "ObjectsDeleted"
- "SelectedChanged"
- "VisibleChanged"
- "StateReset"

The callbackFunction parameter is the actual Python function (not a string) to be called each time the specified change happens.

The return value is an integer callback ID, which can be used to remove the callback later. See `AppRemoveCallback`.

## 1.3 AppCurrentDir

```
std::string AppCurrentDir()
```

Returns the current working directory.

## 1.4 AppCurrentDirSet

```
bool AppCurrentDirSet(const std::string &name)
```

Sets the current working directory and returns whether or not the change was successful.

## 1.5 AppDir

```
std::string AppDir()
```

Returns the path of the directory where the application is installed.

## 1.6 AppDocDir

```
std::string AppDocDir()
```

Returns the path to the directory containing all of the documentation.

## 1.7 AppExampleDir

```
std::string AppExampleDir()
```

Returns the path to the directory containing all of the examples provided with the application.

## 1.8 AppInstallDir

```
std::string AppInstallDir()
```

Returns the path to the directory where VIDA is installed.

## 1.9 AppLastDirGet

```
std::string AppLastDirGet()
```

Returns the path to the last directory that was accessed by VIDA. However, if the user has the preference "Open Current Dir" specified, this function will return an empty string.

## 1.10 AppLastDirSet

```
void AppLastDirSet(const std::string &)
```

Stores the path to the last directory that was accessed by VIDA.

## 1.11 AppLastOpenDirGet

```
std::string AppLastOpenDirGet()
```

Returns the path to the last directory from which a file was opened. However, if the user has the preference "Open Current Dir" specified, this function will return an empty string.

## 1.12 AppLastOpenDirSet

```
void AppLastOpenDirSet(const std::string &)
```

Sets the path to the last directory from which a file was opened.

## 1.13 AppLastSaveDirGet

```
std::string AppLastSaveDirGet()
```

Returns the path to the last directory to which a file was saved. However, if the user has the preference "Open Current Dir" specified, this function will return an empty string.

## 1.14 AppLastSaveDirSet

```
void AppLastSaveDirSet(const std::string &)
```

Sets the path to the last directory to which a file was saved.

## 1.15 AppLicenseFile

```
std::string AppLicenseFile()
```

Returns the path to the license file being used by VIDA.

## 1.16 AppLicenseUpdate

```
void AppLicenseUpdate(bool promptonly)
```

Searches for a new valid license file. If `promptonly` is True, VIDA will prompt the user to specify the location of the license file. If it is False, VIDA will search the expected potential locations for a new valid license file before prompting. If a valid license file is found in one of these locations, that file will be used and the user will not be prompted.

## 1.17 AppOpenUrl

```
void AppOpenUrl(const std::string &url)
```

Opens the specified URL using the most appropriate program for the specified URL. For instance, web pages will be opened with the default web browser and files will be opened with the program that that file type is associated.

## 1.18 AppRemoveCallback

```
void AppRemoveCallback(changeName, unsigned int callbackID)
```

Removes a callback created using the AppAddCallback function. The callbackID argument is the value returned from AppAddCallback, and the changeName string must be the same one that was used in the call to AppAddCallback. See `AppAddCallback`.

## 1.19 AppScriptSave

```
bool AppScriptSave(const std::string &fname)
```

Saves the current scripting history to the specified file.

## 1.20 AppUserDir

```
std::string AppUserDir()
```

Returns the path of the application user directory containing the application preferences, settings, startup script, journal file, and license.

## 1.21 AppVersion

```
std::string AppVersion()
```

Returns a string representation of the complete version number

## 1.22 AppVersionMajor

```
int AppVersionMajor()
```

Returns the first, or major, part of the version number as an integer

## 1.23 AppVersionMinor

```
int AppVersionMinor()
```

Returns the second, or minor, part of the version number as an integer

## 1.24 AppVersionBugFix

```
int AppVersionBugFix()
```

Returns the third, or bug fix, part of the version number as an integer

## 1.25 CopyData

```
void CopyData(const std::string &data)
```

Copies the specified data onto the system clipboard where it can be pasted into other applications.

## 1.26 CopyMolecules

```
void CopyMolecules(const std::vector<OEPropDB::OEKey> &keys)
```

Copies the specified molecules onto the system clipboard where they can be pasted into other applications in a variety of formats.

## 1.27 CopyMoleculesScoped

```
void CopyMoleculesScoped(unsigned int scope = BestScope)
```

Copies all of the molecules in the specified scope onto the system clipboard where they can be pasted into other applications in a variety of formats.

## 1.28 Error

```
void Error(const std::string &err)
```

Raises an exception when called from within a script, otherwise, it reports the specified error message to the status bar.

## 1.29 ErrorDetailsDialog

```
void ErrorDetailsDialog(const std::string &title, const std::string &message,
                        const std::string &error)
```

Launches an error dialog that contain detailed information about the error that occurred.

## 1.30 ExtensionsEdit

```
void ExtensionsEdit()
```

Launches the extensions manager.

## 1.31 IsLicensed

```
bool IsLicensed(const std::string &toolkit)
```

Returns whether or not a valid license exists for the specified toolkit. If a valid license is present, that toolkit can be accessed from Python in VIDA. If a valid license is not present, importing that toolkit in Python will fail and throw an exception.

## 1.32 JournalInit

```
void JournalInit(bool force)
```

For internal use only. This function specifies the script as a "Journal Script" which forces the application to clear its current state and run the script starting from a clean state.

## 1.33 ObservableBool

```
BoolObservable &ObservableBool(const std::string &name, bool create=False)
```

Returns a reference to an observable boolean value.

## 1.34 ObservableFloat

```
FloatObservable &ObservableFloat(const std::string &name, bool create=False)
```

Returns a reference to an observable floating point value.

## 1.35 ObservableInt

```
IntObservable &ObservableInt(const std::string &name, bool create=False)
```

Returns a reference to an observable integer value.

## 1.36 ObservableKey

```
KeyObservable &ObservableKey(const std::string &name, bool create=False)
```

Returns a reference to an observable OEKey object.

## 1.37 ObservableString

```
StringObservable &ObservableString(const std::string &name, bool create=False)
```

Returns a reference to an observable string value.

## 1.38 ObservableUInt

```
UIntObservable &ObservableUInt(const std::string &name, bool create=False)
```

Returns a reference to an observable unsigned integer value.

## 1.39 ObservableUpdate

```
UpdateObservable &ObservableUpdate(const std::string &name, bool create=False)
```

Returns a reference to an observable updater.

## 1.40 ObservableVecFloat

```
FloatVectorObservable &ObservableVecFloat(const std::string &name,
                                          bool create=False)
```

Returns a reference to an observable list of floating point values.

## 1.41 ObservableVecInt

```
IntVectorObservable &ObservableVecInt(const std::string &name,
                                      bool create=False)
```

Returns a reference to an observable list of integer values.

## 1.42 ObservableVecString

```
StringVectorObservable &ObservableVecString(const std::string &name,
                                            bool create=False)
```

Returns a reference to an observable list of string values.

## 1.43 ObservableVecUInt

```
UIntVectorObservable &ObservableVecUInt(const std::string &name,
                                        bool create=False)
```

Returns a reference an observable list of unsigned integer values.

## 1.44 Open

```
std::vector<unsigned int> Open(const std::string &filename,
                               const std::string &type="")
std::vector<unsigned int> Open(const std::vector<std::string> &filenames,
                               const std::string &type="")
```

This function opens the specified file or files. The optional type parameter is the file extension to assume for this file.

## 1.45 OpenState

```
void OpenState(const std::string &filename)
```

This function opens the specified state file. This action will clear the current session, including deleting all the loaded molecules.

## 1.46 PasteMolecules

```
void PasteMolecules()
void PasteMolecules(const std::vector<std::string> &,
                    const std::string &format)
```

Pastes all of the molecules that can be found in the system clipboard into the application. Pasted molecules will appear in a new list called “Pasted”.

## 1.47 PreferenceDump

```
void PreferenceDump()
```

This function prints out all the current preferences.

## 1.48 PreferenceGetBool

```
bool PreferenceGetBool(const std::string &pref, bool def=False)
```

This function returns the current boolean value of the given preference.

## 1.49 PreferenceGetColor

```
OESystem::OEColor PreferenceGetColor(const std::string &pref,
                                     const OESystem::OEColor &def=OESystem::OEWhite)
```

This function returns the current color of the given preference.

## 1.50 PreferenceGetDouble

```
double PreferenceGetDouble(const std::string &pref, double def=0.0)
```

This function returns the double precision number (non-integral) value of the given preference.

## 1.51 PreferenceGetFloat

```
float PreferenceGetFloat(const std::string &pref, float def=0.0f)
```

This function returns the floating point (non-integral) value of the given preference.

## 1.52 PreferenceGetInt

```
int PreferenceGetInt(const std::string &pref, int def=0)
```

This function returns the integer value of the given preference.

## 1.53 PreferenceGetString

```
std::string PreferenceGetString(const std::string &pref,
                                const std::string &def="")
```

This function returns the string value of the given preference.

## 1.54 PreferenceGetVBool

```
std::vector<bool> PreferenceGetVBool(const std::string &pref,
                                     const std::vector<bool> &v=std::vector<bool>())
```

This function returns the vector of boolean values for the given preference.

## 1.55 PreferenceGetVDouble

```
std::vector<double> PreferenceGetVDouble(const std::string &pref,
                                         const std::vector<double> &v=std::vector<double>())
```

This function returns the vector of double precision values for the given preference.

## 1.56 PreferenceGetVFloat

```
std::vector<float> PreferenceGetVFloat(const std::string &pref,
                                       const std::vector<float> &v=std::vector<float>())
```

This function returns the vector of floating point values for the given preference.

## 1.57 PreferenceGetVInt

```
std::vector<int> PreferenceGetVInt(const std::string &pref,
                                   const std::vector<int> &v=std::vector<int>())
```

This function returns the vector of integer values for the given preference.

## 1.58 PreferenceIsBool

```
bool PreferenceIsBool(const std::string &pref)
```

This function returns whether the given preference is a boolean preference.

## 1.59 PreferenceIsColor

```
bool PreferenceIsColor(const std::string &pref)
```

This function returns whether the given preference is a color.

## 1.60 PreferenceIsDouble

```
bool PreferenceIsDouble(const std::string &pref)
```

This function returns whether the given preference is a double precision number.

## 1.61 PreferenceIsFloat

```
bool PreferenceIsFloat(const std::string &pref)
```

This function returns whether the given preference is a floating point number.

## 1.62 PreferenceIsInt

```
bool PreferenceIsInt(const std::string &pref)
```

This function returns whether the given preference is an integer.

## 1.63 PreferenceIsSet

```
bool PreferenceIsSet(const std::string &pref)
```

This function returns whether the given preference has a value.

## 1.64 PreferenceIsString

```
bool PreferenceIsString(const std::string &pref)
```

This function returns whether the given preference is a string.

## 1.65 PreferenceIsVBool

```
bool PreferenceIsVBool(const std::string &pref)
```

This function returns whether the given preference is a list of booleans.

## 1.66 PreferenceIsVDouble

```
bool PreferenceIsVDouble(const std::string &pref)
```

This function returns whether the given preference is a list of double precision numbers.

## 1.67 PreferenceIsVFloat

```
bool PreferenceIsVFloat(const std::string &pref)
```

This function returns whether the given preference is a list of floating point numbers.

## 1.68 PreferenceIsVInt

```
bool PreferenceIsVInt(const std::string &pref)
```

This function returns whether the given preference is a list of floating point numbers.

## 1.69 PreferenceMark

```
void PreferenceMark()
```

This function marks the preferences as a point for undoing.

## 1.70 PreferenceRemove

```
bool PreferenceRemove(const std::string &pref)
```

This function deletes a preference.

## 1.71 PreferenceRestore

```
void PreferenceRestore()
```

This function restores the default preferences.

## 1.72 PreferenceSetBool

```
bool PreferenceSetBool(const std::string &pref, bool v)
```

This function sets the boolean value of the preference.

## 1.73 PreferenceSetColor

```
bool PreferenceSetColor(const std::string &pref, const OESystem::OEColor &v)
```

This function sets the color value of the preference.

## 1.74 PreferenceSetCommand

```
bool PreferenceSetCommand(const std::string &pref, const std::string &cmd)
```

This function sets the command to activate the changes in the preference.

## 1.75 PreferenceSetDouble

```
bool PreferenceSetDouble(const std::string &pref, double v)
```

This function sets double precision value of the preference.

## 1.76 PreferenceSetFloat

```
bool PreferenceSetFloat(const std::string &pref, float v)
```

This function sets floating point value of the preference.

## 1.77 PreferenceSetInt

```
bool PreferenceSetInt(const std::string &pref, int v)
```

This function sets integer value of the preference.

## 1.78 PreferenceSetString

```
bool PreferenceSetString(const std::string &pref, const std::string &v)
```

This function sets the string value of the preference.

## 1.79 PreferenceSetVBool

```
bool PreferenceSetVBool(const std::string &pref, const std::vector<bool> &v)
```

This function sets value of the preference to the passed list of booleans.

## 1.80 PreferenceSetVDouble

```
bool PreferenceSetVDouble(const std::string &pref,
                          const std::vector<double> &v)
```

This function sets value of the preference to the passed list of double precision numbers.

## 1.81 PreferenceSetVFloat

```
bool PreferenceSetVFloat(const std::string &pref,
                         const std::vector<float> &v)
```

This function sets value of the preference to the passed list of floating point numbers.

## 1.82 PreferenceSetVInt

```
bool PreferenceSetVInt(const std::string &pref, const std::vector<int> &v)
```

This function sets value of the preference to the passed list of integers.

## 1.83 PreferenceValidateCommands

```
std::string PreferenceValidateCommands()
```

This function checks all the commands for the preferences to ensure that they all exist. It returns a list of Preference commands that could not be found.

## 1.84 PreferencesEdit

```
void PreferencesEdit()
```

This function opens the preferences dialog.

## 1.85 Quit

```
void Quit(bool force=False)
void quit(bool force=False)
```

Exits the application. If *force* is set to `False`, the user will be prompted before actually exiting.

## 1.86 Redo

```
void Redo()
void Redo(unsigned int steps)
```

This function redoes any previously undone actions.

## 1.87 RedoTo

```
bool RedoTo(const std::string &)
```

Redoes all the previously undone operation up to the specified location in the undo stack.

(See `UndoMark`.)

## 1.88 RunTheGauntlet

```
void RunTheGauntlet()
```

This function is designed for testing and debugging purposes. It is disabled in release distributions.

## 1.89 Save

```
void Save(const std::string &filename, const std::vector<unsigned int> &ids)
void Save(const std::string &filename,
          const std::vector<OEPropDB::OEKey> &keys)
```

This function saves the passed IDs or OEKeys to a file with the given filename.

## 1.90 SaveMiniState

```
void SaveMiniState(const std::string &filename, const std::string &version="")
```

This function saves the current application state as a "mini state" to the specified file.

## 1.91 SaveState

```
void SaveState(const std::string &filename, const std::string &version="")
```

This function saves the current application state to the specified file.

## 1.92 SaveStateFilter

```
void SaveStateFilter(const std::string &filename, const std::string &filter)
```

This function saves the state of the application to the specified file in the state file format determined by the specified *filter*.

## 1.93 SettingsGetBool

```
bool SettingsGetBool(const std::string &s, bool def=False)
```

This function returns the specified boolean setting.

## 1.94 SettingsGetFloat

```
float SettingsGetFloat(const std::string &s, float def=0.0f)
```

This function returns the specified float setting.

## 1.95 SettingsGetInt

```
int SettingsGetInt(const std::string &s, int def=0)
```

This function returns the specified int setting.

## 1.96 SettingsGetString

```
std::string SettingsGetString(const std::string &s, const std::string &def="")
```

This function returns the specified string setting.

## 1.97 SettingsRestore

```
void SettingsRestore()
```

This function returns the specified restore setting.

## 1.98 SettingsSetBool

```
bool SettingsSetBool(const std::string &s, bool v)
```

This function sets the specified bool setting.

## 1.99 SettingsSetFloat

```
bool SettingsSetFloat(const std::string &s, float v)
```

This function sets the specified float setting.

## 1.100 SettingsSetInt

```
bool SettingsSetInt(const std::string &s, int v)
```

This function sets the specified integer setting.

## 1.101 SettingsSetString

```
bool SettingsSetString(const std::string &s, const std::string &v)
```

This function sets the specified string setting.

## 1.102 SettingsSync

```
void SettingsSync()
```

This function synchronizes the application settings with those stored on disk. Ordinarily this function should not need to be called.

## 1.103 StateMark

```
int StateMark()
```

Reserved for internal use.

## 1.104 StatePop

```
int StatePop(bool load=True)
```

Reserved for internal use.

## 1.105 Undo

```
void Undo()
void Undo(unsigned int steps)
```

This function undoes the previous actions.

## 1.106 UndoHint

```
void UndoHint(const std::string &action, bool replace=True)
```

This function is used to name the current undo action. Ordinarily this function should not need to be called by the user.

## 1.107 UndoMark

```
bool UndoMark(const std::string &)
```

Sets a name for the current position in the undo stack which can be used later by the `UndoTo` function.

## 1.108 UndoTo

```
bool UndoTo(const std::string &)
```

Undoes all the previous operations up to the specified operation.

(See `UndoMark`.)

## 1.109 UpdateRedo

```
void UpdateRedo()
```

This function updates the redo menu. Ordinarily this function should not need to be called by the user.

## 1.110 VFSleep

```
void VFSleep(unsigned int seconds)
```

Sleeps the application for the specified number of seconds.

## 1.111 ViewerExportPOVRAY

```
void ViewerExportPOVRAY(const std::string &fname)
```

This function exports the current scene in the 3D display to a POV-Ray input file. The resulting file can be used as input to the POV-Ray raytracing program to generate very high quality static 3D images.

## 1.112 ViewerScreenshot

```
void ViewerScreenshot(const std::string &fname)
void ViewerScreenshot(const std::string &fname, unsigned int width,
                      unsigned int height)
```

This function saves a screenshot of the 3D display window to the specified file (*fname*) in PNG format.

## 1.113 WriteHistory

```
void WriteHistory(const std::string &filename)
```

Writes the current Python interpreter history to the specified filename.

CHAPTER

TWO

# SCOPE FUNCTIONS

The functions detailed in this section enable the user to access information about and change the state of the application through the multiple different available scopes.

## 2.1 Active

```
void Active(const OEPropDB::OEKey &k)
void Active(unsigned int id, unsigned int conf=UINT_MAX)
```

Makes the specified object the *Active* or *Focused* object in the application. The object is specified by passing either an OEKey object or a unique ID paired with a conformer index (if dealing with multiconformer molecules).

## 2.2 ActiveConformer

```
unsigned int ActiveConformer()
```

Returns the index of the currently *Active* or *Focused* conformer.

## 2.3 ActiveID

```
unsigned int ActiveID()
```

Returns the unique ID associated with the currently *Active* or *Focused* object.

## 2.4 ActiveKey

```
OEPropDB::OEKey ActiveKey()
```

Returns the unique key associated with the currently *Active* or *Focused* object.

## 2.5 ClearActive

```
void ClearActive()
```

Clears the *active* / *focused* property from the currently *active* object.

## 2.6 ClearLocked

```
void ClearLocked()
```

Clears the locked property from all objects.

## 2.7 ClearMarked

```
void ClearMarked()
```

Clears the marked property from all objects.

## 2.8 ClearSelection

```
void ClearSelection()
```

Clears the selected property from all objects.

## 2.9 ClearVisible

```
void ClearVisible()
```

Clears the visible property from all objects.

## 2.10 ContextClear

```
void ContextClear()
```

Clears the context scope.

## 2.11 ContextGet

```
std::vector<OEPropDB::OEKey> ContextGet()
```

Returns the list of keys currently contained in the context scope. The "context" scope is used to determine which operations are appropriate for display in the right-click popup menu.

## 2.12 ContextInsert

```
void ContextInsert(const OEPropDB::OEKey &)
void ContextInsert(const std::vector<OEPropDB::OEKey> &)
```

Inserts the specified key(s) into the context scope. The "context" scope is used to determine which operations are appropriate for display in the right-click popup menu.

## 2.13 ContextKeysSet

```
void ContextKeysSet(const OEPropDB::OEKeyGroup &k)
void ContextKeysSet(const std::vector<OEPropDB::OEKey> &keys)
```

Sets the specified keys to be the "context" keys which are used to determine which operations are appropriate for display in the right-click popup menu.

## 2.14 ContextSet

```
void ContextSet(const OEPropDB::OEKey &)
void ContextSet(const std::string &prop)
void ContextSet(const std::vector<OEPropDB::OEKey> &)
```

Sets the specified key(s) to be the "context" keys which are used to determine which operations are appropriate for display in the right-click popup menu.

## 2.15 DefaultScopeGet

```
unsigned int DefaultScopeGet()
```

Returns the default scope of operations.

## 2.16 DefaultScopeSet

```
void DefaultScopeSet(unsigned int scope)
```

Sets the default scope of operations. Valid parameters are:

- **ActiveScope - operations are performed on the *active* / *focused*** object
- **ActiveThenVisibleScope - operations are performed only on the *active* /** *focused* object; however, if no object has that property, the operations are performed on the *visible* objects
- **AllScope - operations are performed on all of the currently loaded** objects
- **BestScope - allows the application to determine the current best scope** for operations based on the context of the operation
- MarkedScope - operations are performed on all of the *marked* objects
- **ScratchScope - operations are performed on all of the objects that were** added to the *scratch* scope using the `ScratchInsert` and the `ScratchSet` functions.
- SelectedScope - operations are performed only on *selected* objects
- VisibleScope - operations are performed on all of the *visible* objects

## 2.17 GetAllContextKeys

```
std::vector<OEPropDB::OEKey> GetAllContextKeys()
```

Returns all of the relevant keys of interest corresponding to the current state of the application.

## 2.18 GetAtomsScoped

```
std::vector<OEPropDB::OEKey> GetAtomsScoped(unsigned int scope)
```

Returns a list of all the atom keys in the specified scope.

## 2.19 GetBondsScoped

```
std::vector<OEPropDB::OEKey> GetBondsScoped(unsigned int scope)
```

Returns a list of all the bond keys in the specified scope.

## 2.20 GetContoursScoped

```
std::vector<OEPropDB::OEKey> GetContoursScoped(unsigned int scope)
```

Returns a list of all the contour keys in the specified scope.

## 2.21 GetGridsScoped

```
std::vector<OEPropDB::OEKey> GetGridsScoped(unsigned int scope)
```

Returns a list of all the grid keys in the specified scope.

## 2.22 GetMarkedAtoms

```
std::vector<OEPropDB::OEKey> GetMarkedAtoms()
```

Returns a list of keys corresponding to all of the currently marked atoms.

## 2.23 GetMarkedAtomsScoped

```
std::vector<OEPropDB::OEKey> GetMarkedAtomsScoped(unsigned int scope)
```

Returns a list of keys corresponding to all of the currently marked atoms that are also in the specified scope.

## 2.24 GetMarkedBonds

```
std::vector<OEPropDB::OEKey> GetMarkedBonds()
```

Returns a list of keys corresponding to all of the currently marked bonds.

## 2.25 GetMarkedBondsScoped

```
std::vector<OEPropDB::OEKey> GetMarkedBondsScoped(unsigned int scope)
```

Returns a list of keys corresponding to all of the currently marked bonds that are also in the specified scope.

## 2.26 GetMarkedContours

```
std::vector<OEPropDB::OEKey> GetMarkedContours()
```

Returns a list of keys corresponding to all of the currently marked contours.

## 2.27 GetMarkedGrids

```
std::vector<OEPropDB::OEKey> GetMarkedGrids()
```

Returns a list of keys corresponding to all of the currently marked grids.

## 2.28 GetMarkedMolecules

```
std::vector<OEPropDB::OEKey> GetMarkedMolecules()
```

Returns a list of keys corresponding to all of the currently marked molecules.

## 2.29 GetMarkedMonitors

```
std::vector<OEPropDB::OEKey> GetMarkedMonitors()
```

Returns a list of keys corresponding to all of the currently marked monitors.

## 2.30 GetMarkedSurfaces

```
std::vector<OEPropDB::OEKey> GetMarkedSurfaces()
```

Returns a list of keys corresponding to all of the currently marked surfaces.

## 2.31 GetMoleculesScoped

```
std::vector<OEPropDB::OEKey> GetMoleculesScoped(unsigned int scope)
```

Returns a list of all the molecule keys in the specified scope.

## 2.32 GetScoped

```
std::vector<OEPropDB::OEKey> GetScoped(unsigned int scope)
std::vector<OEPropDB::OEKey> GetScoped(unsigned int scope, unsigned int type)
```

Returns a list of all the keys in the specified scope. If a *type* parameter is specified, it will return a list of all keys of that type in the specified scope. The type parameter can be retrieved by calling the `GetType` method on an `OEKey` object.

## 2.33 GetSelectedAtom

```
OEPropDB::OEKey GetSelectedAtom()
```

Returns a key corresponding to a single selected atom. If multiple atoms are currently selected, this is equivalent to returning the first entry in list returned by `GetSelectedAtoms`.

## 2.34 GetSelectedAtoms

```
std::vector<OEPropDB::OEKey> GetSelectedAtoms()
```

Returns a list of keys corresponding to the currently selected atoms.

## 2.35 GetSelectedBond

```
OEPropDB::OEKey GetSelectedBond()
```

Returns a key corresponding to a single selected bond. If multiple atoms are currently selected, this is equivalent to returning the first entry in list returned by `GetSelectedBondss`.

## 2.36 GetSelectedBonds

```
std::vector<OEPropDB::OEKey> GetSelectedBonds()
```

Returns a list of keys corresponding to the currently selected bonds.

## 2.37 GetSelectedContours

```
std::vector<OEPropDB::OEKey> GetSelectedContours()
```

Returns a list of keys corresponding to the currently selected contours.

## 2.38 GetSelectedGrids

```
std::vector<OEPropDB::OEKey> GetSelectedGrids()
```

Returns a list of keys corresponding to the currently selected grids.

## 2.39 GetSelectedMolecules

```
std::vector<OEPropDB::OEKey> GetSelectedMolecules()
```

Returns a list of keys corresponding to the currently selected molecules.

## 2.40 GetSelectedMonitors

```
std::vector<OEPropDB::OEKey> GetSelectedMonitors()
```

Returns a list of keys corresponding to the currently selected monitors.

## 2.41 GetSelectedSurfaces

```
std::vector<OEPropDB::OEKey> GetSelectedSurfaces()
```

Returns a list of keys corresponding to the currently selected surfaces.

## 2.42 GetSurfacesScoped

```
std::vector<OEPropDB::OEKey> GetSurfacesScoped(unsigned int scope)
```

Returns a list of all the surface keys in the specified scope.

## 2.43 GetVisibleAtoms

```
std::vector<OEPropDB::OEKey> GetVisibleAtoms()
```

Returns a list of all the visible atom keys.

## 2.44 GetVisibleBonds

```
std::vector<OEPropDB::OEKey> GetVisibleBonds()
```

Returns a list of all the visible bond keys.

## 2.45 GetVisibleContours

```
std::vector<OEPropDB::OEKey> GetVisibleContours()
```

Returns a list of all the visible contour keys.

## 2.46 GetVisibleGrids

```
std::vector<OEPropDB::OEKey> GetVisibleGrids()
```

Returns a list of all the visible grid keys.

## 2.47 GetVisibleIDs

```
std::vector<unsigned int> GetVisibleIDs()
```

Returns a list of all the IDs for the current set of visible objects.

## 2.48 GetVisibleMolecules

```
std::vector<OEPropDB::OEKey> GetVisibleMolecules()
```

Returns a list of all the visible molecule keys.

## 2.49 GetVisibleMonitors

```
std::vector<OEPropDB::OEKey> GetVisibleMonitors()
```

Returns a list of all the visible monitor keys.

## 2.50 GetVisibleSurfaces

```
std::vector<OEPropDB::OEKey> GetVisibleSurfaces()
```

Returns a list of all the visible surface keys.

## 2.51 IDGet

```
unsigned int IDGet(const OESystem::OEBase &)
```

Returns the ID associated with the specified Python object.

## 2.52 IDTypeGet

```
std::string IDTypeGet(unsigned int)
```

Returns the name of the object type associated with the specified ID.

## 2.53 IsActive

```
bool IsActive(const OEPropDB::OEKey &k)
bool IsActive(unsigned int id, unsigned int conf=UINT_MAX)
```

Returns whether or not the specified object (key or ID with optional conformer index) is active/focused.

## 2.54 IsLocked

```
bool IsLocked(const OEPropDB::OEKey &k)
bool IsLocked(unsigned int id, unsigned int conf=UINT_MAX)
```

Returns whether or not the specified object (key or ID with optional conformer index) is locked.

## 2.55 IsMarked

```
bool IsMarked(const OEPropDB::OEKey &k)
bool IsMarked(unsigned int id, unsigned int conf=UINT_MAX)
```

Returns whether or not the specified object (key or ID with optional conformer index) is marked.

## 2.56 IsSelected

```
bool IsSelected(const OEPropDB::OEKey &k)
bool IsSelected(unsigned int id, unsigned int conf=UINT_MAX)
```

Returns whether or not the specified object (key or ID with optional conformer index) is selected.

## 2.57 IsTrulyVisible

```
bool IsTrulyVisible(const OEPropDB::OEKey &k)
bool IsTrulyVisible(unsigned int id, unsigned int conf=UINT_MAX)
```

Returns whether or not the specified object (key or ID with optional conformer index) is truly visible.

## 2.58 IsVisible

```
bool IsVisible(const OEPropDB::OEKey &k)
bool IsVisible(unsigned int id, unsigned int conf=UINT_MAX)
```

Returns whether or not the specified object (key or ID with optional conformer index) is visible.

## 2.59 Lock

```
void Lock(const std::string &k)
void Lock(const OEPropDB::OEKey &k, bool state)
void Lock(const std::vector<OEPropDB::OEKey> &keys, bool state)
void Lock(unsigned int id, bool state, unsigned int conf=UINT_MAX)
```

Sets the *locked* property on the specified object to the specified value (*state*). If a string value is passed to this function as opposed to an object specifier, the *locked* property will be set for all the objects that satisfy the specified query. For more details on the query language, see the relevant chapter in the main VIDA manual.

## 2.60 LockScoped

```
void LockScoped(bool state, unsigned int scope)
```

Sets the *locked* property to the specified value for all objects in the specified scope.

## 2.61 Mark

```
void Mark(const std::string &prop)
void Mark(const OEPropDB::OEKey &k, bool state)
void Mark(const std::vector<OEPropDB::OEKey> &keys, bool state)
void Mark(unsigned int id, bool state, unsigned int conf=UINT_MAX)
void Mark(const std::vector<OEPropDB::OEKey> &keys, bool state,
          unsigned char action)
```

Sets the *marked* property on the specified object to the specified value (*state*). If a string value is passed to this function as opposed to an object specifier, the *marked* property will be set for all the objects that satisfy the specified query. For more details on the query language, see the relevant chapter in the main VIDA manual.

## 2.62 MarkScoped

```
void MarkScoped(bool state, unsigned int scope)
```

Sets the *marked* property to the specified value for all objects in the specified scope.

## 2.63 MimicParentVisibilityGet

```
bool MimicParentVisibilityGet(const OEPropDB::OEKey &k)
```

Returns whether or not the visibility of the specified object is controlled by its parent's visibility.

## 2.64 MimicParentVisibilitySet

```
void MimicParentVisibilitySet(const OEPropDB::OEKey &k, bool mimic)
void MimicParentVisibilitySet(const std::vector<OEPropDB::OEKey> &k,
                              bool mimic)
```

Sets whether or not the visibility of the specified object(s) is controlled by each individual's parent's visibility.

## 2.65 MimicParentVisibilitySetScoped

```
void MimicParentVisibilitySetScoped(const std::string &tag, bool mimic,
                                    unsigned int scope = BestScope)
```

Sets whether or not the visibility of a certain set of objects with the specified scope is controlled by each individual's parent's visibility. The set of objects is determined by the presence or absence of the specified *tag* on the molecule.

## 2.66 ScratchClear

```
void ScratchClear()
```

This function clears the ScratchScope.

## 2.67 ScratchGet

```
std::vector<OEPropDB::OEKey> ScratchGet()
```

Returns all of the keys currently in the ScratchScope.

## 2.68 ScratchInsert

```
void ScratchInsert(const OEPropDB::OEKey &)
void ScratchInsert(const std::vector<OEPropDB::OEKey> &)
```

This function adds a key to the ScratchScope.

## 2.69 ScratchSet

```
void ScratchSet(const std::string &prop)
void ScratchSet(const OEPropDB::OEKey &)
void ScratchSet(const std::vector<OEPropDB::OEKey> &)
```

This function sets the contents of the ScratchScope.

## 2.70 Select

```
void Select(const std::string &prop)
void Select(const OEPropDB::OEKey &k, bool state)
void Select(const std::vector<OEPropDB::OEKey> &keys, bool state)
void Select(unsigned int id, bool state, unsigned int conf=UINT_MAX)
void Select(const std::vector<OEPropDB::OEKey> &keys, bool state,
            unsigned char action)
```

This function selects objects according to the selection language.

## 2.71 SelectAllMolecules

```
void SelectAllMolecules()
```

This function selects every molecule that is currently visible.

## 2.72 SelectByQuery

```
void SelectByQuery(const std::string &query, int action=0,
                   unsigned int scope = BestScope)
```

This function selects every molecule in the specified scope based on whether or not it matches the specified substructure *query*. Valid query inputs include SMILES, SMARTS, and IUPAC as well as common names. The IUPAC and common name to structure conversion is performed by the Ogham toolkit.

## 2.73 SelectInvert

```
void SelectInvert()
```

This function selects the currently visible, but not selected items, and deselects the visible and selected items.

## 2.74 SelectKeys

```
void SelectKeys(const std::vector<OEPropDB::OEKey> &atomsKeys)
```

This function selects the objects corresponding to each of the keys specified in the parameter list *keys*.

## 2.75 SelectOERID

```
void SelectOERID(unsigned int id, bool selected=true)
```

This function sets the selected property on the object corresponding to the specified *id*. The *selected* parameter specifies whether or not the object should either be selected or unselected.

## 2.76 SelectResidues

```
void SelectResidues(const std::vector<OEPropDB::OEKey> &keys, bool state)
```

Set the selected state of the residues containing the specified atoms.

## 2.77 SelectScoped

```
void SelectScoped(bool state, unsigned int scope)
```

Select or unselect (based on the *state* parameter) all the objects in the specified scope.

## 2.78 SelectWithin

```
void SelectWithin(float radius, bool selected)
```

This function selects all molecule objects, including atoms and bonds, within the specified radius of the currently selected objects.

## 2.79 SelectWithout

```
void SelectWithout(float radius, bool selected)
```

This function selects all molecule objects, including atoms and bonds, outside of the specified radii of the currently selected objects.

## 2.80 Subset

```
void Subset(const std::string &name, const std::string &sel)
```

Creates a named subset that can be then used in the Select routine.

## 2.81 Visible

```
void Visible(const std::string &prop)
void Visible(const OEPropDB::OEKey &k, bool state)
void Visible(const std::vector<OEPropDB::OEKey> &keys, bool state)
void Visible(unsigned int id, bool state, unsigned int conf=UINT_MAX)
void Visible(const std::vector<OEPropDB::OEKey> &keys, bool state,
             unsigned char action)
bool Visible(const std::vector<unsigned int> &ids, bool visible,
             unsigned int conf=UINT_MAX)
```

This function sets the keys that pass the specified criteria to be visible.

## 2.82 VisibleScoped

```
void VisibleScoped(bool state, unsigned int scope)
```

Sets the visibility of all the objects in the specified scope based on the *state* parameter.

## 2.83 VisualizeCommandScopeGet

```
unsigned int VisualizeCommandScopeGet()
```

Reserved for future expansion.

## 2.84 VisualizeCommandScopeSet

```
void VisualizeCommandScopeSet(unsigned int scope)
```

Reserved for future expansion.

CHAPTER

THREE

# USER INTERFACE FUNCTIONS

The functions detailed in this section provide access to information about and enable customization of the user interface.

## 3.1 AppComponentNameGet

```
std::string AppComponentNameGet()
```

Returns the name of which VIDA component currently has the mouse focus.

## 3.2 AppComponentNameSet

```
void AppComponentNameSet(const std::string &name)
```

Stores the name of which VIDA component currently has the mouse focus. This function is called internally whenever the mouse focus changes to a new widget within the application. This is primarily intended for internal use and as such should not be called.

## 3.3 AppGeometryGet

```
std::vector<unsigned int> AppGeometryGet()
```

Returns a list of four unsigned integers corresponding to the width, height, x and y positions of the application.

## 3.4 AppGeometrySet

```
std::vector<unsigned int> AppGeometrySet( const std::vector<unsigned int> &geom )
std::vector<unsigned int> AppGeometrySet( unsigned int width,
                                          unsigned int height,
                                          unsigned int x,
                                          unsigned int y,
                                          unsigned int mainx ,
                                          unsigned int mainy )
```

Sets the width, height, x position, y position, central widget width, and central widget height of the application.

## 3.5 AppGlobalFontGet

```
std::string AppGlobalFontGet()
```

Returns the current global font used in the application.

## 3.6 AppGlobalFontSet

```
void AppGlobalFontSet(const std::string &font)
```

Sets the global font to be used in the application.

## 3.7 AppLayoutGet

```
std::string AppLayoutGet()
```

Returns a hexadecimal encoded string representation of the current layout.

## 3.8 AppLayoutSet

```
std::string AppLayoutSet(const std::string &geom)
```

Sets the current layout of the application based on the hexadecimal encoded layout specified by `geom`. Ordinarily this function should not need to be called by the user; however, in the event that it is called, the input to this function should always be the return value from a previous call to `AppLayoutGet`.

## 3.9 AppLayoutToIcon

```
std::vector<std::string> AppLayoutToIcon(unsigned int width  = 24,
                                         unsigned int height = 24 )
```

Returns a cartoon representation of the current layout as an image of size `width` and `height`. The image is returned in XPM format.

## 3.10 AppMainWindowGet

```
std::string AppMainWindowGet()
```

Returns the name of the current main window.

## 3.11 AppMainWindowSet

```
std::string AppMainWindowSet(const std::string &mainWindow)
```

Sets the current main window. The name of the desired main window is the input to this function.

## 3.12 AppMovableWindowsGet

```
bool AppMovableWindowsGet()
```

Returns whether or not the layout of the individual windows in the application can be changed by clicking and dragging the window to the desired location.

## 3.13 AppMovableWindowsSet

```
bool AppMovableWindowsSet(bool)
```

Sets whether or not the layout of the individual windows in the application can be changed by clicking and dragging the window to the desired location.

## 3.14 AppPerspectiveSet

```
std::string AppPerspectiveSet(const std::string &perspective)
```

Sets the current layout of the application based on the hexadecimal encoded layout specified by `perspective` while maintaining the current geometry (width, height, x and y positions). Ordinarily this function should not need to be called by the user; however, in the event that it is called, the input to this function should always be the return value of a previous call to `AppLayoutGet`.

## 3.15 AppRecordWindowEventsGet

```
bool AppRecordWindowEventsGet()
```

Returns whether or not window events are currently being recorded in the application history.

## 3.16 AppRecordWindowEventsSet

```
bool AppRecordWindowEventsSet(bool record)
```

Sets whether or not window events are recorded in the application history.

## 3.17 AppShowGet

```
std::string AppShowGet()
```

Returns the display status of the application. Potential return values are:

- “normal”
- “minimized”
- “maximized”
- “fullscreen”

## 3.18 AppShowMenuBarGet

```
bool AppShowMenuBarGet()
```

Returns whether or not the main menubar is currently visible.

## 3.19 AppShowMenuBarSet

```
void AppShowMenuBarSet(bool)
```

Sets whether or not the main menubar is currently visible.

## 3.20 AppShowSet

```
void AppShowSet(const std::string &style)
```

Sets the display status of the application. Valid options are:

- "normal"
- "minimized"
- "maximized"
- "fullscreen"

## 3.21 AppShowStatusBarGet

```
bool AppShowStatusBarGet()
```

Returns whether or not the main status bar is currently visible.

## 3.22 AppShowStatusBarSet

```
void AppShowStatusBarSet(bool)
```

Sets whether or not the main status bar is currently visible.

## 3.23 AppSloppyFocusGet

```
bool AppSloppyFocusGet()
```

Returns whether or not the window focus is changed by simply positioning the mouse over a window as opposed to requiring a click in the window to change focus (the default behavior).

## 3.24 AppSloppyFocusSet

```
bool AppSloppyFocusSet(bool sloppy)
```

Sets whether or not the window focus is changed by simply positioning the mouse over a window as opposed to requiring a click in the window to change focus (the default behavior).

## 3.25 AppStatusTextSet

```
void AppStatusTextSet(const std::string &text, unsigned int seconds)
```

Sets the displayed text in the status bar of the main application window to the specified text. If the *seconds* parameter is 0, this text will replace the default SMILES display in the status bar. If the *seconds* parameter is greater than 0, the text will only be displayed for the specified period of time, returning to the default SMILES display when the time has elapsed.

## 3.26 AppWindowStyleGet

```
std::string AppWindowStyleGet()
```

Returns the application window display style of the application. These styles typically correspond to specific operating system window managers.

## 3.27 AppWindowStyleSet

```
std::string AppWindowStyleSet(const std::string &windowStyle)
```

Sets the application window display style of the application. These styles typically correspond to specific operating system window managers.

## 3.28 BlockBegin

```
void BlockBegin()
```

Blocks user interaction with the GUI with the exception of the abort button. This function is used internally and should only be called by the user to restore already existing blocking behavior that may have been temporarily overridden to enable user interaction within a user specified script.

## 3.29 BlockEnd

```
void BlockEnd()
```

Restores user interaction with the GUI that was previously blocked by `BlockBegin`. This function is used internally and should only be called to enable user interaction (if needed) within a user specified script. Any script which calls this function should restore the blocking behavior by calling `BlockBegin` once the user interaction portion is complete. However, it is important to make sure that blocking is actually active in the first place (see `BlockGet`) before calling this function and then later calling `BlockBegin`.

## 3.30 BlockExemptGet

```
bool BlockExemptGet(const std::string &name)
```

Returns whether or not the GUI object specified by *name* is exempt from GUI blocking when Python commands are being executed.

## 3.31 BlockExemptSet

```
void BlockExemptSet(const std::string &name, bool value)
```

Sets whether or not the GUI object specified by *name* is exempt from GUI blocking when Python commands are being executed.

## 3.32 BlockGet

```
bool BlockGet()
```

Returns whether or not user interaction with the GUI is currently being blocked. User interaction is typically blocked while Python commands and scripts are being executed.

## 3.33 BuilderFocusOnProperties

```
void BuilderFocusOnProperties()
```

Displays the *Properties* tab in the Builder Window.

## 3.34 BuilderFocusOnSketcher

```
void BuilderFocusOnSketcher()
```

Displays the *Sketcher* tab in the Builder Window.

## 3.35 IconGetXPM

```
std::vector<std::string> IconGetXPM(const std::string &name)
```

Returns an XPM representation of the icon specified by name.

## 3.36 InterpreterClear

```
void InterpreterClear()
```

Clears the contents of the scripting window.

## 3.37 InterpreterPopUpdateGUI

```
void InterpreterPopUpdateGUI()
```

```
Specifies to the application that the normal interface updating behavior
should occur after issued commands. This function is normally called in
conjunction with
:oe:func:`InterpreterPushUpdateGUI` to
terminate the non-updating behavior triggered by that function. It is
important to note that these functions can be nested.
```

## 3.38 InterpreterPushUpdateGUI

```
void InterpreterPushUpdateGUI(bool update)
```

Specifies to the application whether or not the application should be updating the interface after this function is called. This is normally called to allow multiple commands to be issued in a row without causing an interface update after each individual call. A call to `InterpreterPopUpdateGUI` will resume the normal updating behavior. It is important to note that these calls can be nested.

## 3.39 InterpreterShowDebugTextGet

```
bool InterpreterShowDebugTextGet()
```

Returns whether or not debugging text will be displayed in the scripting window.

## 3.40 InterpreterShowDebugTextSet

```
void InterpreterShowDebugTextSet(bool show)
```

Sets whether or not debugging text will be displayed in the scripting window.

## 3.41 InterpreterTimerGet

```
bool InterpreterTimerGet()
```

Returns whether or not the application reports the time elapsed after every user entered command.

## 3.42 InterpreterTimerSet

```
bool InterpreterTimerSet(bool enable)
```

Sets whether or not the application reports the time elapsed after every user entered command.

## 3.43 InterpreterUpdatesGUI

```
bool InterpreterUpdatesGUI()
```

Returns whether or not VIDA is updating the interface after every command is completed.

## 3.44 LayoutCurrentGet

```
std::string LayoutCurrentGet()
```

Returns the name of the most recently applied named layout.

## 3.45 LayoutExists

```
bool LayoutExists(const std::string &name)
```

Returns whether or not the specified named layout exists.

## 3.46 LayoutGet

```
std::string LayoutGet(const std::string &name)
```

Returns a string representation of the specified name layout. The value returned by this function can be used as input to the following functions:

- AppLayoutSet
- AppPerspectiveSet

## 3.47 LayoutIcon

```
std::vector<std::string> LayoutIcon(const std::string &name)
```

Returns the icon associated with the specified named layout in XPM format.

## 3.48 LayoutLoad

```
bool LayoutLoad(const std::string &name)
```

Loads the specified named layout.

## 3.49 LayoutOrganizePrompt

```
std::string LayoutOrganizePrompt()
```

Launches the layout organization dialog.

## 3.50 LayoutOverrideOrder

```
void LayoutOverrideOrder(const std::vector<std::string> &names)
```

Sets the order to be used when displaying the named layouts in the layout button.

## 3.51 LayoutRemove

```
bool LayoutRemove(const std::string &name)
```

Removes the specified named layout.

## 3.52 LayoutRename

```
bool LayoutRename(const std::string &oldName, const std::string &newName)
```

Renames the specified named layout.

## 3.53 LayoutSaveCurrent

```
bool LayoutSaveCurrent(const std::string &name)
```

Saves the current application's layout using the specified name.

## 3.54 LayoutSet

```
void LayoutSet(const std::string &name, const std::string &data)
```

Creates a named layout using the specified name and layout data. The input for the *data* parameter can be obtained from `AppLayoutGet` command.

## 3.55 LayoutStickyGet

```
bool LayoutStickyGet()
```

## 3.56 LayoutStickySet

```
void LayoutStickySet(bool sticky)
```

## 3.57 Layouts

```
std::vector<std::string> Layouts()
```

Returns a list of the available named layouts.

## 3.58 ListWindowCollapseAll

```
void ListWindowCollapseAll()
```

Collapses all the visible nodes in the List Window to show only the top-level list entries.

## 3.59 ListWindowCollapseCurrent

```
void ListWindowCollapseCurrent()
```

Collapses the currently active list node in the List Window to a single top-level entry.

## 3.60 ListWindowExpandAll

```
void ListWindowExpandAll()
```

Expands all the top-level list nodes in the List Window to show their contents.

## 3.61 ListWindowExpandCurrent

```
void ListWindowExpandCurrent()
```

Expands the currently active list node in the List Window to show its contents.

## 3.62 ListWindowFirstList

```
void ListWindowFirstList()
```

Makes active the first item in the first list.

## 3.63 ListWindowFirstListItem

```
void ListWindowFirstListItem()
```

Makes active the first item in the current list.

## 3.64 ListWindowFocusOnOERID

```
void ListWindowFocusOnOERID(unsigned int id)
```

Changes the view of the list window to ensure that the specified ID is visible.

## 3.65 ListWindowGetCurrentPos

```
std::vector<unsigned int> ListWindowGetCurrentPos()
```

Returns the current active position with the list window. This function is primarily intended for internal use only.

## 3.66 ListWindowHideColumn

```
void ListWindowHideColumn(unsigned int column)
```

Hides the specified column from view in the list window.

## 3.67 ListWindowIsVisible

```
bool ListWindowIsVisible()
```

Returns whether or not the list window is currently visible.

## 3.68 ListWindowLastList

```
void ListWindowLastList()
```

Makes active the first item in the last list.

## 3.69 ListWindowLastListItem

```
void ListWindowLastListItem()
```

Makes active the last item in the current list.

## 3.70 ListWindowNavigateChildrenGet

```
bool ListWindowNavigateChildrenGet()
```

Returns whether or not to navigate through an object's children when browsing through the list window.

## 3.71 ListWindowNavigateChildrenSet

```
void ListWindowNavigateChildrenSet(bool nav)
```

Sets whether or not to navigate through an object's children when browsing through the list window.

## 3.72 ListWindowNavigateLeft

```
void ListWindowNavigateLeft()
```

Navigate to a new active object as if the user had pressed the left arrow key inside of the list window.

## 3.73 ListWindowNavigateRight

```
void ListWindowNavigateRight()
```

Navigate to a new active object as if the user had pressed the right arrow key inside of the list window.

## 3.74 ListWindowNextList

```
bool ListWindowNextList()
```

Makes active the first item in the next list.

## 3.75 ListWindowNextListItem

```
bool ListWindowNextListItem()
```

Makes active the next item in the current list.

## 3.76 ListWindowPrevList

```
bool ListWindowPrevList()
```

Makes active the first item in the previous list.

## 3.77 ListWindowPrevListItem

```
bool ListWindowPrevListItem()
```

Makes active the previous item in the current list.

## 3.78 ListWindowRowColoringGet

```
bool ListWindowRowColoringGet()
```

Returns whether or not the list window uses an alternating row coloring scheme.

## 3.79 ListWindowRowColoringSet

```
void ListWindowRowColoringSet(bool on)
```

Sets whether or not the list window uses an alternating row coloring scheme.

## 3.80 ListWindowSetCurrentPos

```
void ListWindowSetCurrentPos(std::vector<unsigned int> pos)
```

Sets the current active position within the list window. This is intended for internal use only.

## 3.81 ListWindowShowColumn

```
void ListWindowShowColumn(unsigned int column)
```

Shows the specified column in the list window.

## 3.82 MainWindowScreenshot

```
bool MainWindowScreenshot(const std::string &filename="")
```

Captures an image of the current main window and saves that image to the specified file.

## 3.83 MainWindowScreenshotPrompt

```
bool MainWindowScreenshotPrompt()
```

Opens a dialog which allows the user to specify multiple parameters regarding the creation of a screenshot of the main window.

## 3.84 MenuAddButton

```
std::string MenuAddButton(const std::string &menu, const std::string &button,
                          const std::string &command, bool last=false)
std::string MenuAddButton(const std::string &menu, const std::string &button,
                          const std::string &command, const std::string &hotkey,
                          bool last)
```

Adds a button to the specified menu which when pushed will execute the specified Python command. An optional `hotkey` for this menu button may be provided. The `last` argument, if specified, indicates whether or not this button should always appear at the end of the menu regardless of how many items are added to the menu after this.

Returns a unique name identifier which can be used to reference this button in other functions.

## 3.85 MenuAddRadioButton

```
std::string MenuAddRadioButton(const std::string &menu,
                               const std::string &button,
                               const std::string &command,
                               const std::string &updateCommand)
```

Adds a radio button to the specified menu which when pushed will execute the specified Python command. The `updateCommand` parameter is a Python command which is used to update the checked state of this button.

This function should only be called in between calls to `MenuBeginRadioGroup` and `MenuEndRadioGroup`.

Returns a unique name identifier which can be used to reference this button in other functions.

## 3.86 MenuAddSeparator

```
std::string MenuAddSeparator(const std::string &menu,
                             const std::string &name="",
                             bool last=false,
                             bool conditional=false)
```

Adds a simple line separator to the specified menu. The `last` parameter specifies whether to keep this separator at the end of the menu. The `conditional` parameter specifies whether or not to collapse multiple adjacent separators into a single separator.

Returns a unique name identifier which can be used to reference this separator in other functions.

## 3.87 MenuAddSubmenu

```
std::string MenuAddSubmenu( const std::string &parent,
                            const std::string &menu,
                            bool last = false )
std::string MenuAddSubmenu( const std::string &parent,
                            const std::string &menu,
                            const std::string &displayname,
                            bool last)
std::string MenuAddSubmenu( const std::string &parent,
                            const std::string &menu,
                            const std::string &displayName,
                            const std::string &update,
                            bool last)
```

Adds a submenu to the specified `parent` menu. The `menu` parameter is the desired name for this menu. The `displayName` parameter, if specified, contains the actual text to be shown when this menu is displayed. If no `displayName` parameter is specified, the `menu` parameter will be used. The `update` parameter specifies a Python command which will be called immediately before showing the menu. This command can be used to update the state

of the menu or to generate the menu on the fly as desired. The `last` parameter specifies whether to keep this submenu at the end of the parent menu.

Returns a unique name identifier which can be used to reference this menu in other functions. Often this value will be the same as the value specified by the `menu` parameter, but it is not guaranteed to be the same due to potential name clashes.

## 3.88 MenuAddToggleButton

```
std::string MenuAddToggleButton(const std::string &menu,
                                const std::string &button,
                                const std::string &command,
                                const std::string &updateCommand,
                                bool last=false)
```

Adds a toggle button to the specified menu. Clicking on the button executes the Python code specified in the `command` parameter. The state of the toggle button is available to the Python command in the variable `OEInternal.MenuVal` as either 0 or 1. The `updateCommand` parameter specifies Python code which is called to determine the current state of the button. If this code returns a True value, the toggle will be set to on, if not, it will be set to off.

Returns a unique name identifier which can be used to reference this button in other functions.

## 3.89 MenuBeginRadioGroup

```
  void MenuBeginRadioGroup(const std::string &menu)
```

```
Begins a group of mutually exclusive radio buttons in the specified menu.
```

## 3.90 MenuButtonActionSet

```
void MenuButtonActionSet(const std::string &menu, const std::string &item,
                         const std::string &action)
```

Sets the Python command *action* to be executed by the specified button *item* in the specified menu *menu*.

## 3.91 MenuDisplayName

```
void MenuDisplayName(const std::string &menu, const std::string &item,
                     const std::string &display_name)
```

Sets the display name (i.e. the text actually shown for this menu item) for the specified menu item (button or submenu) in the specified menu.

## 3.92 MenuDynamicSet

```
void MenuDynamicSet(const std::string &menu, const std::string &cmd,
                    bool dynamic)
```

Sets whether or not the specified menu is dynamically updated or not. If the menu is set to be dynamically updated, the Python code contained in the `cmd` parameter will be executed immediately before showing the menu each time the menu is shown.

## 3.93 MenuEnableItem

```
void MenuEnableItem(const std::string &menu, const std::string &item,
                    bool enabled)
```

Sets whether or not the specified menu item is enabled.

## 3.94 MenuEnableItemCommand

```
void MenuEnableItemCommand(const std::string &menu, const std::string &item,
                           const std::string &command)
```

Sets a Python command which is run when a menu is updated to set whether the specified menu item is enabled or not.

## 3.95 MenuEndRadioGroup

```
void MenuEndRadioGroup(const std::string &menu)
```

Ends a group of mutually exclusive radio buttons in the specified menu.

## 3.96 MenuExists

```
bool MenuExists(const std::string &menu)
```

Returns whether or not the specified menu exists.

## 3.97 MenuGetAllItemsContainingName

```
std::vector<std::string> MenuGetAllItemsContainingName(const std::string &name)
```

Searches for all menu items that contain the string “name.”

## 3.98 MenuGetAllItems

```
   std::vector<std::string> MenuGetAllItems(const std::string &id)

Returns a vector of ids of all sub items of a particular menu.
If id is empty then it returns every menu item.
```

## 3.99 MenuHasItem

```
bool MenuHasItem(const std::string &menu, const std::string &item)
```

Returns whether or not the specified menu contains the specified menu item.

## 3.100 MenuItemIsAMenu

```
bool MenuItemIsAMenu ( const std::string &id )
```

Returns true if the id belongs to a menu as opposed to an action menu item.

## 3.101 MenuRemoveAll

```
void MenuRemoveAll(const std::string &menu)
```

Removes the entire contents of the specified menu.

## 3.102 MenuRemoveItem

```
void MenuRemoveItem(const std::string &item)
void MenuRemoveItem(const std::string &menu, const std::string &item)
```

Removes the specified menu item from the specified menu.

## 3.103 MenuUpdateItem

```
void MenuUpdateItem(const std::string &menu, const std::string &item)
```

Updates the specified menu item in the specified menu. Calling this function on a menu item will cause the associated item to execute any previously bound update command to determine the current state of the item.

## 3.104 Pick

```
void Pick(unsigned int action, const std::string &source,
          const std::vector<OEPropDB::OEKey> &keys, int replicate)
void Pick(unsigned int action, unsigned int startX, unsigned int startY,
          unsigned int endX, unsigned int endY, unsigned int width,
          unsigned int height)
```

Picks a small rectangular region on the 3D display specified by the start and end coordinates. This function is primarily intended for internal use.

## 3.105 PickAgain

```
void PickAgain(unsigned int action)
```

Repeats the previous Pick action.

## 3.106 PopIgnoreHint

```
void PopIgnoreHint()
```

This function, paired with PushIgnoreHint, minimizes application response to the functions called between them. Every PushIgnoreHint must be matched with a PopIgnoreHint and vice versa.

## 3.107 PopupAddButton

```
std::string PopupAddButton(const std::string &button,
                           const std::string &command, bool showOnce=false)
```

This function adds a clickable menu entry that runs the specified command to the popup menu. The last optional parameter specifies that this menu entry should only last one show of the popup menu.

## 3.108 PopupAddRadioButton

```
std::string PopupAddRadioButton(const std::string &button,
                                const std::string &command,
                                const std::string &updateCommand,
                                bool showOnce=false)
```

This function adds an exclusive clickable menu entry to the popup menu. The last optional parameter specifies that this menu entry should only last one show of the popup menu. It should only be called between PopupBeginRadioGroup and PopupEndRadioGroup.

## 3.109 PopupAddSeparator

```
std::string PopupAddSeparator(bool once=false, bool conditional=false)
```

This function adds a non-clickable line menu entry to the popup menu. The last optional parameter specifies that this menu entry should only last one show of the popup menu.

## 3.110 PopupAddSetupFunc

```
void PopupAddSetupFunc(const std::string &function)
```

This function specifies a worker function to be called before the popup menu is shown. It is called with no arguments. It is designed to allow context sensitive menu entries to be added to the popup menu.

## 3.111 PopupAddSubmenu

```
std::string PopupAddSubmenu(const std::string &child, bool showOnce=false)
```

This function adds a submenu to the popup menu. The last optional parameter specifies that this menu entry should only last one show of the popup menu.

## 3.112 PopupAddToggleButton

```
std::string PopupAddToggleButton(const std::string &button,
                                 const std::string &command,
                                 const std::string &updateCommand,
                                 bool showOnce=false, bool state=false)
```

This function adds a clickable menu entry that runs the specified command to the popup menu. The last optional parameter specifies that this menu entry should only last one show of the popup menu. The state of the toggle is passed as either 0 or 1 in the variable OEInternal.MenuVal. The return value of the update command is used to set the state of the toggle. Returning 1 or True from the update function sets the toggle on.

## 3.113 PopupBeginRadioGroup

```
void PopupBeginRadioGroup()
```

This function specifies that the following added radio buttons are exclusive.

## 3.114 PopupDisplayName

```
void PopupDisplayName(const std::string &item, const std::string &display_name)
```

This function modifies the display name of a popup menu entry.

## 3.115 PopupEnableItem

```
void PopupEnableItem(const std::string &item, bool enabled)
```

This function enables or disables the specified popup menu entry.

## 3.116 PopupEnableItemCommand

```
void PopupEnableItemCommand(const std::string &item,
                            const std::string &command)
```

This function sets the update command on the given entry in the popup menu.

## 3.117 PopupEndRadioGroup

```
void PopupEndRadioGroup()
```

This function ends the radio group begun by PopupBeginRadioGroup.

## 3.118 PopupRemoveAll

```
void PopupRemoveAll()
```

This function removes all entries from the popup menu.

## 3.119 PopupRemoveItem

```
void PopupRemoveItem(const std::string &item)
```

This function removes the specified entry from the popup menu.

## 3.120 ProgressbarUpdate

```
bool ProgressbarUpdate(int count, int total)
```

Updates the application progress bar based on the specified *count* and *total* parameters.

## 3.121 PromptAtomicNum

```
unsigned int PromptAtomicNum(const std::string &caption, unsigned int def=6)
```

Prompts the user to specify an element from a periodic table dialog.

## 3.122 PromptColor

```
OEColor PromptColor(const OEColor &deflt=OEColor())
```

Prompts the user to specify a color.

## 3.123 PromptError

```
bool PromptError(const std::string &)
```

This function prompts the user with an error message.

## 3.124 PromptFilename

```
std::string PromptFilename(const std::string &def="",
                           const std::string &mode="Open",
                           const std::string &type="all",
                           const std::string &msg="")
```

This function prompts the user for a filename. The parameters specify the default value, whether it should be an 'Open' or 'Save' dialog, and what kind of file.

## 3.125 PromptFilenames

```
std::vector<std::string>
  PromptFilenames(const std::vector<std::string> &def=std::vector<std::string>(),
                  const std::string &type="all", const std::string &msg="")
```

This function prompts the user for a collection of filenames. The parameters specify the default value, whether it should be an 'Open' or 'Save' dialog, and what kind of files.

## 3.126 PromptFloat

```
float PromptFloat(const std::string &prompt, float def=0.0f,
                  unsigned int prec=5)
```

This function prompts the user for a floating point number.

## 3.127 PromptFont

```
std::string PromptFont()
```

Prompts the user to specify a desired font.

## 3.128 PromptID

```
unsigned int PromptID(unsigned int filters=0xF0FF, bool preview=true)
unsigned int PromptID(std::vector<std::string> filterList, bool preview=true)
```

This function prompts the user for an ID from the currently loaded set.

Unsigned integer filters are an OR'ed combination of the following values.

- PromptFilter_Molecules
- PromptFilter_Surfaces
- PromptFilter_Grids
- PromptFilter_Lists
- PromptFilter_Reflections
- PromptFilter_Selected

- PromptFilter_Active
- PromptFilter_Marked
- PromptFilter_Visible
- PromptFilter_Unknown
- PromptFilter_All

String filters are just placed in a list:

- 'M' or 'm' for molecules
- 'S' or 's' for surfaces
- 'R' or 'r' for reflections
- 'G' or 'g' for grids
- 'L' or 'l' for lists
- 'O' or 'o' for Unknown objects

For example, to prompt for Molecules and Surfaces:

id = PromptID(PromptFilter_Molecules | PromptFilter_Surfaces)

or

id = PromptID(['M','S'])

## 3.129 PromptIDWriteFilters

```
std::vector<std::string> PromptIDWriteFilters(const std::string &fname)
```

This function returns the valid types that can be written to the type of file specified by the filename parameter.

## 3.130 PromptIDs

```
std::vector<unsigned int> PromptIDs(unsigned int filters=0xF0FF,
                                    bool preview=true)
std::vector<unsigned int> PromptIDs(const std::vector<std::string> &filterList,
                                    bool preview=true)
```

This function prompts the user for several IDs from the currently loaded set.

See *PromptID* for the list of available filters.

## 3.131 PromptInteger

```
int PromptInteger(const std::string &msg, int def=0)
```

This function prompts the user for an integer.

## 3.132 PromptKey

```
OEPropDB::OEKey PromptKey(unsigned int filters=0xF0FF, bool preview=true)
OEPropDB::OEKey PromptKey(const std::vector<std::string> &filterList,
                          bool preview=true)
```

This function prompts the user for a key.

See *PromptID* for the list of available filters.

## 3.133 PromptKeys

```
std::vector<OEPropDB::OEKey>
  PromptKeys(const std::vector<std::string> &filterList, bool preview=true)
std::vector<OEPropDB::OEKey> PromptKeys(unsigned int filters=0xF0FF,
                                        bool preview=true)
```

This function prompts the user for a collection of key.

See *PromptID* for the list of available filters.

## 3.134 PromptMessage

```
bool PromptMessage(const std::string &msg)
```

This function alerts the user with a message and an OK button.

## 3.135 PromptModeGet

```
std::string PromptModeGet()
```

## 3.136 PromptModeSet

```
bool PromptModeSet(const std::string &mod)
```

## 3.137 PromptFragment

```
std::string PromptMolecule(const std::string &caption)
```

Prompts the user to input a molecule fragment using the built-in molecule input dialog. Returns a hexadecimal encoded OEB string representation of the fragment. In addition to specifying a molecular fragment, the user will be required to define an attachment point, either by including a dummy atom or a selected atom or bond.

## 3.138 PromptMolecule

```
std::string PromptMolecule(const std::string &caption,
                           const std::string &def="", unsigned int options=21)
```

Prompts the user to specify a molecule using the built-in molecule input dialog. Returns a hexadecimal encoded OEB string representation of the molecule.

## 3.139 PromptMoleculeSplit

```
std::string PromptMoleculeSplit(unsigned int id)
```

Launches a dialog to assist in the splitting of the specified molecule.

## 3.140 PromptMulti

```
bool PromptMulti(const std::string &prompt, std::vector<std::string> names,
                 std::vector<std::string> types,
                 std::vector<std::string> defs,
                 std::vector<OEInterpreter::OEMultiTypeVar> &out)
```

This function prompts the user for several different types of variables and returns them in a list.

## 3.141 PromptQuery

```
std::string PromptQuery(const std::string &def="")
```

This function prompts the user for a query string for calls to `ListSubsetQuery`.

## 3.142 PromptResponseBool

```
void PromptResponseBool(const std::string &msg, bool v, bool forceInter)
```

This function pushes a boolean response to an upcoming Prompt, allowing it to be skipped in scripts. Ordinarily this function should not need to be called by the user.

## 3.143 PromptResponseCanceled

```
void PromptResponseCanceled(const std::string &)
```

This function pushes a cancel response to an upcoming Prompt, allowing it to be skipped in scripts. Ordinarily this function should not need to be called by the user.

## 3.144 PromptResponseColor

```
void PromptResponseColor(const std::string &type, const OESystem::OEColor &c,
                         bool forceInteractive)
```

This function pushes a color response to an upcoming Prompt, allowing it to be skipped in scripts. Ordinarily this function should not need to be called by the user.

## 3.145 PromptResponseFloat

```
void PromptResponseFloat(const std::string &type, float v, bool forceInter)
```

This function pushes a floating point response to an upcoming Prompt, allowing it to be skipped in scripts. Ordinarily this function should not need to be called by the user.

## 3.146 PromptResponseInt

```
void PromptResponseInt(const std::string &type, int v, bool forceInter=false)
```

This function pushes an integer response to an upcoming Prompt, allowing it to be skipped in scripts. Ordinarily this function should not need to be called by the user.

## 3.147 PromptResponseKey

```
void PromptResponseKey(const std::string &type, const OEPropDB::OEKey &key,
                       bool forceInteractive=false)
```

This function pushes a key response to an upcoming Prompt, allowing it to be skipped in scripts. Ordinarily this function should not need to be called by the user.

## 3.148 PromptResponseKeys

```
void PromptResponseKeys(const std::string &type,
                        std::vector<OEPropDB::OEKey> &keys,
                        bool forceInteractive=false)
```

This function pushes a collection of keys response to an upcoming Prompt, allowing it to be skipped in scripts. Ordinarily this function should not need to be called by the user.

## 3.149 PromptResponseString

```
void PromptResponseString(const std::string &type, const std::string &v,
                          bool forceInteractive=false)
```

This function pushes a string response to an upcoming Prompt, allowing it to be skipped in scripts. Ordinarily this function should not need to be called by the user.

## 3.150 PromptResponseUInt

```
void PromptResponseUInt(const std::string &type, unsigned int v,
                        bool forceInteractive=false)
```

This function pushes a positive integer response to an upcoming Prompt, allowing it to be skipped in scripts. Ordinarily this function should not need to be called by the user.

## 3.151 PromptResponseVectInt

```
void PromptResponseVectInt(const std::string &type,
                           const std::vector<int> &value,
                           bool forceInteractive=false)
```

This function pushes a list-of-integers response to an upcoming Prompt, allowing it to be skipped in scripts. Ordinarily this function should not need to be called by the user.

## 3.152 PromptResponseVectMulti

```
void PromptResponseVectMulti(const std::string &type,
                             const std::vector<OEInterpreter::OEMultiTypeVar> &value,
                             bool forceInteractive=false)
```

## 3.153 PromptResponseVectString

```
void PromptResponseVectString(const std::string &type,
                              const std::vector<std::string> &value,
                              bool forInteractive=false)
```

This function pushes a list-of-strings response to an upcoming Prompt, allowing it to be skipped in scripts. Ordinarily this function should not need to be called by the user.

## 3.154 PromptResponseVectUInt

```
void PromptResponseVectUInt(const std::string &type,
                            const std::vector<unsigned int> &value,
                            bool forceInteractive=false)
```

This function pushes a list-of-positive-integers response to an upcoming Prompt, allowing it to be skipped in scripts. Ordinarily this function should not need to be called by the user.

## 3.155 PromptSaveFilename

```
std::string PromptSaveFilename(const std::vector<unsigned int> &ids,
                               const std::string &msg="")
std::string PromptSaveFilename(const std::vector<OEPropDB::OEKey> &keys,
```

```
                                  const std::string &msg="")
std::vector<std::string> PromptSaveFilename(const std::string &filters,
                                            const std::string &msg="")
```

This function prompts the user to select a file for saving. The *filters* parameter allows for specification of filter types to be shown in the file dialog. Individual filters should be separated by a pair of semicolons like this:

"OpenEye OEBinary (*.oeb);;All Files (*.*)"

The *msg* parameter allows for specification of the caption of the file dialog. This function returns a list where the first member is the selected filename and the second is the selected filter. If the user cancels the prompt, the returned list will be empty.

See *PromptID* for the list of available filters.

## 3.156 PromptSmiles

```
std::string PromptSmiles(const std::string &caption, const std::string &def="",
                         unsigned int options=21)
```

Prompts the user to specify a molecule using the built-in molecule input dialog. The molecule is returned as a SMILES string.

## 3.157 PromptString

```
std::string PromptString(const std::string &prompt="",
                         const std::string &def="", bool lineMode=true,
                         bool editable=true)
```

This function prompts the user for a string.

## 3.158 PromptStringListFromString

```
void PromptStringListFromString(const std::string &in,
                                std::vector<std::string> &options,
                                unsigned int &def)
```

This function prompts the user for a string.

## 3.159 PromptStringListToString

```
std::string PromptStringListToString(const std::vector<std::string> &options,
                                     unsigned int def)
std::string PromptStringListToString(const std::vector<std::string> &options,
                                     const std::string &def)
```

This function generates a list of strings from a formatted string. Ordinarily this function should not need to be called by the user.

## 3.160 PromptYesNo

```
bool PromptYesNo(const std::string &prompt, const std::string &key="")
bool PromptYesNo(const std::string &prompt, bool &check,
                 const std::string &key="")
```

This function prompts the user for a yes or no answer.

## 3.161 PromptYesNoSetDefault

```
bool PromptYesNoSetDefault(const std::string &key, bool def)
```

This function prompts the user for a yes or no answer.

## 3.162 PushIgnoreHint

```
void PushIgnoreHint(bool ignore)
```

This function delays update actions for the following commands until a PopIgnoreHint is called. PushIgnoreHint and PopIgnoreHints MUST match.

## 3.163 SetProgressbar

```
void SetProgressbar(int index, int total)
```

This function sets the progress bar to the specified value.

## 3.164 ToolbarAdd

```
void ToolbarAdd(const std::vector<std::string> &icon,
                const std::string &command, const std::string &tooltip="",
                const std::string &toolbar="Application",
                const std::string &name="")
void ToolbarAdd(const std::string &text, const std::string &command,
                const std::string &tooltip="",
                const std::string &toolbar="Application",
                const std::string &name="", int margin=5)
```

Adds a single standard push button to the specified toolbar. The button can be represented by either the passed icon (in XPM format) or the passed text string. The specified command will be executed when the button is pushed.

## 3.165 ToolbarAddAbort

```
void ToolbarAddAbort(const std::string &toolbar)
```

Adds an abort button to the specified toolbar.

## 3.166 ToolbarAddCombo

```
void ToolbarAddCombo(const std::vector<std::string> &choices,
                     const std::vector<std::string> &commands,
                     const std::string &update,
                     const std::string &toolbar="Application",
                     const std::string &name="")
```

Adds a combo box (a.k.a. dropdown box) with the specified list of options and their associated commands to the specified toolbar. The *update* parameter contains a command that will be executed to update the widget.

## 3.167 ToolbarAddMenu

```
void ToolbarAddMenu(const std::vector<std::string> &text,
                    const std::vector<std::string> &command,
                    const std::vector<std::string> &tooltip,
                    const std::string &update,
                    const std::string &toolbar="Application",
                    const std::string &name="", int margin=5)
void ToolbarAddMenu(const std::vector<std::vector<std::string> > &icon,
                    const std::vector<std::string> &text,
                    const std::vector<std::string> &command,
                    const std::vector<std::string> &tooltip,
                    const std::string &update,
                    const std::string &toolbar="Application",
                    const std::string &name="")
```

Adds a button with a dropdown menu to the toolbar. Each menu item is represented by a name, a Python command, and a tooltip. One implementation also allows specification of an icon for each item. The *update* parameter contains a command that will be executed to update the button and the associated menu.

## 3.168 ToolbarAddMenuViaNamedIcons

```
void ToolbarAddMenuViaNamedIcons(const std::vector<std::string> &iconName,
                                 const std::vector<std::string> &text,
                                 const std::vector<std::string> &command,
                                 const std::vector<std::string> &tooltip,
                                 const std::string &update,
                                 const std::string &toolbar="Application",
                                 const std::string &name="")
```

Adds a button with a dropdown menu to the toolbar. Each menu item is represented by a name, a named icon, a Python command, and a tooltip. The *update* parameter contains a command that will be executed to update the button and the associated menu.

## 3.169 ToolbarAddSeparator

```
void ToolbarAddSeparator(const std::string &toolbar="Application")
```

Adds a separator to the specified toolbar.

## 3.170 ToolbarAddSheet

```
StyleSheet *ToolbarAddSheet(const std::string &icon,
                            const std::string &toolbar)
```

Adds a new style sheet to the specified toolbar represented by the specified icon.

## 3.171 ToolbarAddSlider

```
void ToolbarAddSlider(const std::string &applyCommand,
                      const std::string &getCount, const std::string &getMin,
                      const std::string &getMax, const std::string &getCurrent,
                      const std::string &sliderToVal,
                      const std::string &valToSlider,
                      const std::string &toolbar="Application",
                      const std::string &name="")
```

Adds a slider bar to the specified toolbar. The *applyCommand* will be executed whenever the slider bar is changed. The *getCount* parameter is a command that returns how many things the slider is currently controlling. The *getMin*, *getMax*, and *getCurrent* parameters are commands that will be executed to determine the bounds of the slider. The *sliderToVal* parameter is a command that converts the slider position to a meaningful value for the *applyCommand* function. The *valToSlider* parameter is a command that converts the value returned by *getCurrent* to a meaningful value to be shown on the slider bar.

## 3.172 ToolbarAddToggle

```
void ToolbarAddToggle(const std::string &onText, const std::string &offText,
                      const std::string &onCommand,
                      const std::string &offCommand, const std::string &desc,
                      const std::string &update,
                      const std::string &toolbar="Application",
                      const std::string &name="", int margin=5)
void ToolbarAddToggle(const std::vector<std::string> &onIcon,
                      const std::vector<std::string> &offIcon,
                      const std::string &onCommand,
                      const std::string &offCommand, const std::string &desc,
                      const std::string &update,
                      const std::string &toolbar="Application",
                      const std::string &name="")
```

Adds a single standard toggle button to the specified toolbar. Both on and off icons can be specified as parameters (in XPM format). Commands for both the on and off states can be specified and will be executed when the associated button enters that state.

## 3.173 ToolbarAddToggleViaNamedIcons

```
void ToolbarAddToggleViaNamedIcons(const std::string &onIconName,
                                   const std::string &ofIconfName,
                                   const std::string &onCommand,
                                   const std::string &offCommand,
                                   const std::string &desc,
```

```
                                    const std::string &update,
                                    const std::string &toolbar="Application",
                                    const std::string &name="")
```

Equivalent to ToolbarAddToggle except that instead of having to pass in an icon pixmaps, icons are specified by name. For a table of available icons, please see the Icons chapter in the main VIDA manual.

## 3.174 ToolbarAddViaNamedIcon

```
void ToolbarAddViaNamedIcon(const std::string &iconName,
                            const std::string &command,
                            const std::string &tooltip="",
                            const std::string &toolbar="Application",
                            const std::string &name="")
```

Equivalent to ToolbarAdd except that instead of having to pass in an icon pixmap, the icon is specified by name. For a table of available icons, please see the Icons chapter in the main VIDA manual.

## 3.175 ToolbarBeginRadio

```
  void ToolbarBeginRadio(const std::string &name,
                         const std::string &toolbar="Application")
```

```
Begins a named group of mutually exclusive radio buttons in the specified
toolbar.
```

## 3.176 ToolbarButtonEnableGet

```
bool ToolbarButtonEnableGet(const std::string &name,
                            const std::string &toolbar="Application")
```

Returns whether or not the specified button in the specified toolbar is currently enabled.

## 3.177 ToolbarButtonEnableSet

```
void ToolbarButtonEnableSet(const std::string &name, bool enabled,
                            const std::string &toolbar="Application")
```

Sets whether or not the specified button in the specified toolbar is currently enabled.

## 3.178 ToolbarComboAddChoice

```
bool ToolbarComboAddChoice(const std::string &choice,
                           const std::string &command,
                           const std::string &after,
                           const std::string toolbar="Application",
                           const std::string &combo_name="")
```

Adds a new option specified by *choice* to the specified combo box in the specified toolbar. The *command* parameter contains the Python function to be called when this choice is selected. The *after* parameter specifies after which other choice this new option should be added.

## 3.179 ToolbarComboClear

```
bool ToolbarComboClear(const std::string toolbar="Application",
                       const std::string &combo_name="")
```

Clears the contents of the specified combo box in the specified toolbar.

## 3.180 ToolbarComboRemoveChoice

```
bool ToolbarComboRemoveChoice(const std::string &choice,
                              const std::string toolbar="Application",
                              const std::string &combo_name="")
```

Removes the specified *choice* from the specified combo box in the specified toolbar.

## 3.181 ToolbarCreate

```
void ToolbarCreate(const std::string &toolbar)
```

Creates a new toolbar with the specified name.

## 3.182 ToolbarEnabledGet

```
bool ToolbarEnabledGet(const std::string &toolbar="Application")
```

Returns whether or not the specified *toolbar* is currently enabled.

## 3.183 ToolbarEnabledSet

```
bool ToolbarEnabledSet(bool vis, const std::string &toolbar="Application")
```

Sets whether or not the specified *toolbar* is currently enabled.

## 3.184 ToolbarEndRadio

```
void ToolbarEndRadio(const std::string &name,
                     const std::string &toolbar="Application")
```

Ends the named group of mutually exclusive radio buttons in the specified toolbar.

## 3.185 ToolbarGetAll

```
std::vector<std::string> ToolbarGetAll()
```

Returns a list of all the existing toolbars in VIDA.

## 3.186 ToolbarItemCheckedGet

```
bool ToolbarItemCheckedGet(const std::string &name, const std::string &toolbar)
```

Returns whether or not the specified toolbar item in the specified toolbar is currently checked.

## 3.187 ToolbarItemCheckedSet

```
bool ToolbarItemCheckedSet(bool, const std::string &name,
                           const std::string &toolbar)
```

Sets whether or not the specified toolbar item in the specified toolbar is currently checked.

## 3.188 ToolbarItemUpdate

```
void ToolbarItemUpdate(const std::string &name, const std::string &toolbar)
```

Causes the specified toolbar item in the specified toolbar to update itself.

## 3.189 ToolbarItemVisibleGet

```
bool ToolbarItemVisibleGet(const std::string &name, const std::string &toolbar)
```

Returns whether or no the specified toolbar item in the specified toolbar is currently visible.

## 3.190 ToolbarItemVisibleSet

```
bool ToolbarItemVisibleSet(bool, const std::string &name,
                           const std::string &toolbar)
```

Sets whether or not the specified toolbar item in the specified toolbar is currently visible.

## 3.191 ToolbarRemove

```
void ToolbarRemove(const std::string &toolbar)
void ToolbarRemove(const std::string &toolbar, const std::string &name)
```

Removes the specified *toolbar* from VIDA. Please not that when a toolbar is removed, it is deleted from the application and not simply hidden.

## 3.192 ToolbarUpdate

```
void ToolbarUpdate(const std::string &toolbar)
```

Updates the specified toolbar.

## 3.193 ToolbarVisibleGet

```
bool ToolbarVisibleGet(const std::string &toolbar="Application")
```

Returns whether or not the specified toolbar is currently visible.

## 3.194 ToolbarVisibleSet

```
bool ToolbarVisibleSet(bool vis, const std::string &toolbar="Application")
```

Sets whether or not the specified toolbar is currently visible.

## 3.195 UpdateStyleWidget

```
void UpdateStyleWidget()
```

This function notifies the style widget that it should update itself due to changes that might affect its display.

## 3.196 UpdateUndo

```
void UpdateUndo()
```

This function updates the undo menu. Ordinarily this function should not need to be called by the user.

## 3.197 ViewerActiveAnnotation

```
OEGLTextBox *ViewerActiveAnnotation(int x=-1, int y=-1, int w=-1, int h=-1)
```

Create the active annotation.

## 3.198 ViewerActiveAnnotationBackgroundColorGet

```
OESystem::OEColor ViewerActiveAnnotationBackgroundColorGet()
```

Return the background color for the active annotation.

## 3.199 ViewerActiveAnnotationBackgroundColorSet

```
void ViewerActiveAnnotationBackgroundColorSet(const OESystem::OEColor &clr)
```

Set the background color for the active annotation.

## 3.200 ViewerActiveAnnotationClose

```
void ViewerActiveAnnotationClose()
```

Close the active annotation window.

## 3.201 ViewerActiveAnnotationFontGet

```
std::string ViewerActiveAnnotationFontGet()
```

Return the font used for the active annotation.

## 3.202 ViewerActiveAnnotationFontSet

```
void ViewerActiveAnnotationFontSet(const std::string &font)
```

Set the font used for the active annotation. Uses fonts as returned by the `PromptFont` scripting command.

## 3.203 ViewerActiveAnnotationForegroundColorGet

```
OESystem::OEColor ViewerActiveAnnotationForegroundColorGet()
```

Returns the foreground (text) color used in the active annotation.

## 3.204 ViewerActiveAnnotationForegroundColorSet

```
void ViewerActiveAnnotationForegroundColorSet(const OESystem::OEColor &clr)
```

Sets the foreground (text) color for the active annotation.

## 3.205 ViewerActiveAnnotationVisible

```
bool ViewerActiveAnnotationVisible()
```

Returns whether or not the active annotation is visible.

## 3.206 ViewerActiveDataFontSizeGet

```
unsigned int ViewerActiveDataFontSizeGet()
```

Returns the font size for the active data display.

## 3.207 ViewerActiveDataFontSizeSet

```
void ViewerActiveDataFontSizeSet(unsigned int sz)
```

Sets the font size for the active data display.

## 3.208 ViewerActiveDataHide

```
OEGLDataView *ViewerActiveDataHide()
```

Hides the active data display.

## 3.209 ViewerActiveDataShow

```
OEGLDataView *ViewerActiveDataShow()
OEGLDataView *ViewerActiveDataShow(int x, int y, int h, int w)
```

Returns and shows the active data display.

## 3.210 ViewerActiveDataVisible

```
bool ViewerActiveDataVisible()
```

Returns whether or not the active data display is currently visible in the 3D display.

## 3.211 ViewerBookmarkWidget

```
OEGLHList *ViewerBookmarkWidget()
```

Returns a reference to the bookmark control widget.

## 3.212 ViewerBookmarkWidgetFontSizeGet

```
unsigned int ViewerBookmarkWidgetFontSizeGet()
```

Returns the size of the font used in the bookmark control widget.

## 3.213 ViewerBookmarkWidgetFontSizeSet

```
void ViewerBookmarkWidgetFontSizeSet(unsigned int sz)
```

Sets the size of the font used in the bookmark control widget.

## 3.214 ViewerBookmarkWidgetHide

```
OEGLHList *ViewerBookmarkWidgetHide()
```

Hides the bookmark control widget from the 3D display.

## 3.215 ViewerBookmarkWidgetShow

```
OEGLHList *ViewerBookmarkWidgetShow()
```

Shows the bookmark control widget in the 3D display.

## 3.216 ViewerButtonImage

```
OEGLButton *ViewerButtonImage(const std::string &image,
                              const std::string &callback,
                              int x, int y, int w=-1, int h=-1)
```

Creates and returns a clickable button in the 3D display. The *image* parameter specifies the name of the icon to be displayed on the button. The *callback* parameter specifies the Python function to be executed when the button is clicked on. The position and dimensions of the button are specified by the parameters *x*, *y*, *w*, and *h*.

## 3.217 ViewerClick

```
void ViewerClick(int fxn, int x, int y, int gX, int gY, int w, int h)
```

This function performs an action in the display corresponding to a mouse click event in the 3D display window. Ordinarily, this function should not need to be called by the user.

## 3.218 ViewerCursorSet

```
void ViewerCursorSet(const std::string &mode)
```

This function sets the cursor style based on the *mode* parameter. Valid *mode* parameters are:

- arrow - sets the cursor to the standard arrow cursor
- back diagonal arrow - sets the cursor to a backwards diagonal double-ended arrow
- blank - sets the cursor to a blank cursor (no cursor will appear)
- busy - sets the cursor to be a busy indicator (an hourglass on many systems)

- closed hand - sets the cursor to be a closed hand
- cross - sets the cursor to two crossed lines
- double arrow - sets the cursor to two crossed double-ended arrows
- forbidden - sets the cursor to be a circle with a cross line through it
- forward diagonal arrow - sets the cursor to a forwards diagonal double-ended arrow
- horizontal arrow - sets the cursor to a horizontal double-ended arrow
- ibeam - sets the cursor to an ibeam style cursor
- open hand - sets the cursor to be an open hand
- pointing - sets the cursor to be a pointing hand
- rotation - same as *cross*
- split horizontal - sets the cursor to two parallel horizontal lines with perpendicular arrows coming out of each line
- split vertical - sets the cursor to two parallel vertical lines with perpendicular arrows coming out of each line
- translation - same as *double arrow*
- up arrow - sets the cursor to an upwards point arrow
- vertical arrow - sets the cursor to a vertical double-ended arrow
- wait - sets the cursor to a waiting indicator (an hourglass on many systems)
- whats this - sets the cursor to be an arrow with an adjacent question mark

## 3.219 ViewerDepict

```
OEGLImage *ViewerDepict(const OEPropDB::OEKey &key, int x, int y, int width,
                        int height)
```

Creates and returns a depiction widget for the 3D display. The *key* parameter specifies which molecule to use for the depiction. The position and dimensions of the widget are specified by the *x*, *y*, *width*, and *height* parameters.

## 3.220 ViewerDepictionAntiAliasGet

```
bool ViewerDepictionAntiAliasGet()
```

Returns whether or not the lines used in drawing depictions in the 3D display should be anti-aliased.

## 3.221 ViewerDepictionAntiAliasSet

```
void ViewerDepictionAntiAliasSet(bool s)
```

Sets whether or not the lines used in drawing depictions in the 3D display should be anti-aliased.

## 3.222 ViewerDepictionHeightGet

```
float ViewerDepictionHeightGet()
```

Returns the percent of the total height of the 3D display that the active depiction display uses when drawing onto the 3D display.

## 3.223 ViewerDepictionLineWidthGet

```
float ViewerDepictionLineWidthGet()
```

Returns the width of the lines used in drawing depictions in the 3D display.

## 3.224 ViewerDepictionLineWidthSet

```
void ViewerDepictionLineWidthSet(float lw)
```

Sets the width of the lines used in drawing depictions in the 3D display.

## 3.225 ViewerDepictionSizeSet

```
float ViewerDepictionSizeSet(float percent)
float ViewerDepictionSizeSet(float perc_w, float perc_h)
```

Sets the percents of the total width and height of the 3D display that the active depiction display uses when drawing onto the 3D display. The *perc_w* parameter specifies the percent width and the *perc_h* specifies the percent height.

## 3.226 ViewerDepictionWidthGet

```
float ViewerDepictionWidthGet()
```

Returns the percent of the total width of the 3D display that the active depiction display uses when drawing onto the 3D display.

## 3.227 ViewerLabel

```
OEGLLabel *ViewerLabel(const std::string &message, int x, int y)
```

Creates and returns a text label which is displayed in the 3D window. The text is specified by the *message* parameter and the position is specified by the *x* and *y* parameters.

## 3.228 ViewerLabelDialog

```
void ViewerLabelDialog(const std::string &type)
```

Opens a dialog which allows the user to specify what types of labels to apply to either atoms or bonds.

## 3.229 ViewerMouseClick

```
void ViewerMouseClick(unsigned int state, unsigned int action)
```

This function specifies the desired *action* that should occur when a mouse click event is generated with the associated *state*. The *state* parameter specifies which mouse buttons were clicked as well as whether any modifier keys were depressed at the same time. *State* parameters can be added together to specify complicated states. Valid *state* parameters are:

- MouseState_None
- MouseState_Left
- MouseState_Middle
- MouseState_Right
- MouseState_Shift
- MouseState_Control
- MouseState_Alt

*Action* parameters are distinct and cannot be added together like *state* parameters. Valid *action* parameters are:

- MouseClick_None
- MouseClick_Menu - creates a context specific popup menu
- **MouseClick_Select - selects the object that was clicked on and clears the** previous selection
- **MouseClick_SelectAdd - selects the object that was clicked on, but does not** clear the previous selection
- **MouseClick_SelectGrow - extends the selection of the object that was** clicked on: atom -> residue -> chain -> mol and clears the previous selection
- **MouseClick_SelectAddGrow - extends the selection of the object that was** clicked on: atom -> residue -> chain -> mol and clears the previous selection
- **MouseClick_SelectToggle - toggles the selection of the object that was** clicked on
- **MouseClick_Label - displays a context specific label corresponding to the** object that was clicked on

## 3.230 ViewerMouseDoubleClick

```
void ViewerMouseDoubleClick(unsigned int state, unsigned int action)
```

This function specifies the desired *action* that should occur when a mouse double-click event is generated with the associated *state*. The *state* parameter specifies which mouse buttons were clicked as well as whether any modifier keys were depressed at the same time. *State* parameters can be added together to specify complicated states. Valid *state* parameters are:

- MouseState_None
- MouseState_Left

- MouseState_Middle
- MouseState_Right
- MouseState_Shift
- MouseState_Control
- MouseState_Alt

*Action* parameters are distinct and cannot be added together like *state* parameters. Valid *action* parameters are:

- MouseClick_None
- MouseClick_Menu - creates a context specific popup menu
- **MouseClick_Select - selects the object that was clicked on and clears the** previous selection
- **MouseClick_SelectAdd - selects the object that was clicked on, but does not** clear the previous selection
- **MouseClick_SelectGrow - extends the selection of the object that was** clicked on: atom -> residue -> chain -> mol and clears the previous selection
- **MouseClick_SelectAddGrow - extends the selection of the object that was** clicked on: atom -> residue -> chain -> mol and clears the previous selection
- **MouseClick_SelectToggle - toggles the selection of the object that was** clicked on
- **MouseClick_Label - displays a context specific label corresponding to the** object that was clicked on

## 3.231 ViewerMouseFunctionGet

```
unsigned int ViewerMouseFunctionGet()
```

Returns the current function of the mouse when used in the 3D display. For more details on the different types of functions, please see the documentation for ViewerMouseFunctionSet.

## 3.232 ViewerMouseFunctionNameGet

```
std::string ViewerMouseFunctionNameGet(unsigned int)
```

Returns the name of the specified mouse function.

## 3.233 ViewerMouseFunctionSet

```
void ViewerMouseFunctionSet(unsigned int action)
void ViewerMouseFunctionSet(const std::string &)
```

Sets the current function of the mouse when used in the 3D display. Valid functions are:

- MouseClick_None - this tells the mouse to act according to behaviors specified by the current mouse map (this is the default behavior)
- MouseClick_Label - this mode causes a detailed information label to appear on the screen regarding whatever is currently under the cursor
- MouseClick_MeasureAngle - this puts the mouse into angle measurement mode

- MouseClick_MeasureDistance - this puts the mouse into distance measurement mode
- MouseClick_MeasureTorsion - this puts the mouse into torsion measurement mode

## 3.234 ViewerMouseMap

```
void ViewerMouseMap(const std::string &map)
```

This function sets the current mouse mapping. Valid parameters include:

- vida
- afitt
- coot
- insight
- moe
- o
- quanta
- rasmol
- sybyl
- maestro

## 3.235 ViewerMouseMove

```
void ViewerMouseMove(unsigned int state, unsigned int action,
                     unsigned int loc=OEGUI::MouseLocation::Both,
                     unsigned int dir=OEGUI::MouseDirection::Both)
```

This function specifies the desired *action* that should occur when a mouse movement event is generated with the associated *state*, in the specified *location*, and in the specified *direction*. The *state* parameter specifies which mouse buttons were clicked as well as whether any modifier keys were depressed at the same time. *State* parameters can be added together to specify complicated states. Valid *state* parameters are:

- MouseState_None
- MouseState_Left
- MouseState_Middle
- MouseState_Right
- MouseState_Shift
- MouseState_Control
- MouseState_Alt

The *location* parameter specifies whether the mouse was inside or outside a central circular region in the 3D display. The bounds of this region can be displayed using the ViewerShowTrackballGuideSet scripting command. Valid *location* parameters are:

- MouseLocation_None
- MouseLocation_Inside

- MouseLocation_Outside
- MouseLocation_Both

The *direction* parameter specifies which direction of mouse motion should control the action. Valid *direction* parameters are:

- MouseDirection_None
- MouseDirection_X
- MouseDirection_Y
- MouseDirection_Both

*Action* parameters are distinct and cannot be added together like *state* parameters. Valid *action* parameters are:

- MouseMotion_None
- MouseMotion_RotateX - rotates the display along the X axis
- MouseMotion_RotateY - rotates the display along the Y axis
- MouseMotion_RotateZ - rotates the display along the Z axis
- MouseMotion_Trackball - rotates the display using a trackball motion
- MouseMotion_TranslateXY - translates the display along the X and Y axes
- MouseMotion_TranslateX - translates the display along the X axis
- MouseMotion_TranslateY - translates the display along the Y axis
- MouseMotion_TranslateZ - translates the display along the Z axis
- MouseMotion_Zoom - zooms the display
- **MouseMotion_Clip - controls the position of the far and near clipping** planes in a mirrored fashion
- MouseMotion_ClipFar - controls the position of the far clipping plane
- MouseMotion_ClipNear - controls the position of the near clipping plane
- **MouseMotion_DepthcueBegin - controls the position of the start of the** depthcueing calculation
- **MouseMotion_DepthcueEnd - controls the position of the end of the** depthcueing calculation
- **MouseMotion_SelectRect - draws a rectangle on the screen and will select** the entire contents of that rectangle when the mouse button is released while clearing the previous selection
- **MouseMotion_SelectRectAdd - draws a rectangle on the screen and will select** the entire contents of that rectangle when the mouse button is released while keeping the previous selection
- **MouseMotion_Label - displays a context specific label containing** information about whatever object is currently beneath the mouse
- **MouseMotion_Contour - adjusts the contour level of the grid contours in the** current scope
- **MouseMotion_TextScale - adjusts the size of the text displayed in the 3D** display

## 3.236 ViewerMouseOutsideAwareGet

```
bool ViewerMouseOutsideAwareGet()
```

Returns whether or not the display window recognizes mouse motion along the outside edge of the window as a potentially different action from mouse motion in the interior of the window. Ordinarily, this function should not need to be called by the user.

## 3.237 ViewerMouseOutsideAwareSet

```
bool ViewerMouseOutsideAwareSet(bool aware)
```

Sets whether or not the display window recognizes mouse motion along the outside edge of the window as a potentially different action from mouse motion in the interior of the window. Ordinarily, this function should not need to be called by the user.

## 3.238 ViewerMouseReset

```
void ViewerMouseReset()
```

Clears the current mouse mapping. Ordinarily, this function should not need to be called by the user. Furthermore, calling this function without rebuilding the mouse map will disable all interaction with the 3D window.

## 3.239 ViewerMouseSensitivityGet

```
float ViewerMouseSensitivityGet()
```

Returns the sensitivity of mouse event detection in the 3D window.

## 3.240 ViewerMouseSensitivitySet

```
float ViewerMouseSensitivitySet(float s)
```

Sets the sensitivity of mouse event detection in the 3D window.

## 3.241 ViewerMouseWheel

```
void ViewerMouseWheel(unsigned int state, unsigned int action)
```

This function specifies the desired *action* that should occur when a mouse wheel event is generated with the associated *state*. The *state* parameter specifies whether any modifier keys were depressed at the same time. *State* parameters can be added together to specify complicated states. Valid *state* parameters are:

- MouseState_None
- MouseState_Shift
- MouseState_Control
- MouseState_Alt

*Action* parameters are distinct and cannot be added together like *state* parameters. Valid *action* parameters are:

- MouseMotion_None

- MouseMotion_RotateX - rotates the display along the X axis
- MouseMotion_RotateY - rotates the display along the Y axis
- MouseMotion_RotateZ - rotates the display along the Z axis
- MouseMotion_TranslateX - translates the display along the X axis
- MouseMotion_TranslateY - translates the display along the Y axis
- MouseMotion_TranslateZ - translates the display along the Z axis
- MouseMotion_Zoom - zooms the display
- **MouseMotion_Clip - controls the position of the far and near clipping** planes in a mirrored fashion
- MouseMotion_ClipFar - controls the position of the far clipping plane
- MouseMotion_ClipNear - controls the position of the near clipping plane
- **MouseMotion_DepthcueBegin - controls the position of the start of the** depthcueing calculation
- **MouseMotion_DepthcueEnd - controls the position of the end of the** depthcueing calculation
- **MouseMotion_Contour - adjusts the contour level of the grid contours in the** current scope
- **MouseMotion_TextScale - adjusts the size of the text displayed in the 3D** display

## 3.242 ViewerMove

```
void ViewerMove(int fxnX, int fxnY, int x, int y, int oldx, int oldy, int w, int h,
                bool first)
```

This function performs an action in the display window corresponding to a mouse move event in the 3D display window. Ordinarily, this function should not need to be called by the user.

## 3.243 ViewerPickSurfaceTrianglesGet

```
bool ViewerPickSurfaceTrianglesGet()
```

Returns whether or not picking on a surface picks individual triangles as opposed to picking the entire surface. Ordinarily, this function should not need to be called by the user.

## 3.244 ViewerPickSurfaceTrianglesSet

```
bool ViewerPickSurfaceTrianglesSet(bool enable)
```

Sets whether or not picking on a surface picks individual triangles as opposed to picking the entire surface. Ordinarily, this function should not need to be called by the user.

## 3.245 ViewerProgress

```
OEGLProgress *ViewerProgress(const std::string &message)
```

Creates and returns a progress bar display shown in the 3D window. The *message* parameter specifies the text associated with the progress bar.

## 3.246 ViewerRemoveWidget

```
void ViewerRemoveWidget(OEGLWidget *widget, bool update)
```

This function removes the specified widget from the 3D display. The *update* parameter specifies whether or not all the other widgets in the 3D display need to be updated as a result of this change.

## 3.247 ViewerTextBox

```
OEGLTextBox *ViewerTextBox(const OEPropDB::OEKey &key, int x=-1, int y=-1,
                           int w=-1, int h=-1)
OEGLTextBox *ViewerTextBox(const std::string &message, int x=-1, int y=-1,
                           int w=-1, int h=-1)
```

Creates and returns a text display widget to be shown in the 3D display. The parameter *message* specifies the current text. The position and dimensions of the widget are specified by the parameters *x*, *y*, *w*, and *h*.

## 3.248 ViewerUseInertiaGet

```
bool ViewerUseInertiaGet()
```

Returns whether or not the 3D display window tracks the inertia of mouse movements and uses that to keep the scene spinning after the mouse button is released.

## 3.249 ViewerUseInertiaSet

```
bool ViewerUseInertiaSet(bool state)
```

Sets whether or not the 3D display window tracks the inertia of mouse movements and uses that to keep the scene spinning after the mouse button is released.

## 3.250 ViewerUseKeyMapGet

```
bool ViewerUseKeyMapGet()
```

Returns whether or not the viewer is using its own internal key bindings. Ordinarily, this function does not need to be called by the user.

## 3.251 ViewerUseKeyMapSet

```
bool ViewerUseKeyMapSet(bool state)
```

Sets whether or not the viewer is using its own internal key bindings. Ordinarily, this function does not need to be called by the user.

## 3.252 ViewerWheel

```
void ViewerWheel(int fxn, int x, int y, int delta)
```

This function performs an action in the display window corresponding to a mouse wheel event in the 3D display window. Ordinarily, this function should not need to be called by the user.

## 3.253 ViewerWidgetUpdate

```
void ViewerWidgetUpdate()
```

Forces an update of all the display widgets in the 3D display.

## 3.254 WaitBegin

```
void WaitBegin()
```

This function sets the application in a waiting state. It is used for long operations. Ordinarily this function should not need to be called by the user.

## 3.255 WaitEnd

```
void WaitEnd()
```

This function stops the application in a waiting state. It is used for long operations. Ordinarily this function should not need to be called by the user.

## 3.256 WindowEnabledGet

```
bool WindowEnabledGet(const std::string &windowName)
```

Returns whether or not the specified window is enabled for use within VIDA.

## 3.257 WindowEnabledSet

```
void WindowEnabledSet(const std::string &windowName, bool enabled)
```

Sets whether or not the specified window is enabled for use within VIDA.

## 3.258 WindowMenuUpdate

```
void WindowMenuUpdate()
```

Updates the contents of the main *Window* menu.

## 3.259 WindowRegisterHotkey

```
void WindowRegisterHotkey(const std::string &windowName,
                          const std::string &keystroke,
                          const std::string &pyCommand)
```

Registers the specified keystroke with the associated Python command in the specified window. If this keystroke is detected when the specified window has the application focus, the associated command will be run.

## 3.260 WindowVisibleGet

```
bool WindowVisibleGet(const std::string &windowName)
```

Returns whether or not the specified window is currently visible.

## 3.261 WindowVisibleSet

```
bool WindowVisibleSet(const std::string &windowName, bool visible,
                      const std::string &pos="")
```

Sets whether or not the specified window is currently visible.

CHAPTER

FOUR

# DISPLAY FUNCTIONS

The functions detailed in this section enable the customization of how the application itself as well as the currently loaded objects are displayed.

## 4.1 AnnotationGet

```
std::string AnnotationGet(const OEPropDB::OEKey &key)
```

Returns the text annotation associated with the object specified by `key`.

## 4.2 AnnotationHas

```
bool AnnotationHas(const OEPropDB::OEKey &key)
```

Returns whether or not the object specified by `key` has a text annotation.

## 4.3 AnnotationSet

```
void AnnotationSet(const OEPropDB::OEKey &key, const std::string &txt,
                   bool notify=true)
```

Sets a text annotation on the object specified by `key` containing the text specified by `txt`. If `notify` is set to `True`, VIDA will be notified to update the user interface to reflect this change.

## 4.4 AtomClearColorScoped

```
void AtomClearColorScoped(unsigned int scope = BestScope)
```

This function restores the default atom coloring to those atoms within the specified scope.

## 4.5 AtomColorPaletteUpdate

```
void AtomColorPaletteUpdate()
```

This function updates the default atom coloring palette based on whether or not the application is using a dark or light colored background scheme. This function is called internally and should not be called by the user.

## 4.6 AtomColorReferenceScoped

```
void AtomColorReferenceScoped(unsigned int scope = BestScope)
```

This function provides a quick way to mark a molecule (or set of atoms) as a reference structure. The coloring function that is applied colors all carbon atoms the current application 'reference' color (default is green). The 'reference' color can be changed in the preferences.

## 4.7 AtomColorResidueScoped

```
void AtomColorResidueScoped(unsigned int scope = BestScope)
```

This function provides a quick way to color a molecule (or set of atoms) based on associated residue information. The default coloring that is applied uses the 'Shapely' color scheme for coloring residues. The residue colors can be changed in the preferences:

```
* ALA     Medium Green  [140,255,140]
* GLY     White         [255,255,255]
* LEU     Olive Green   [ 69, 94, 69]
* SER     Medium Orange [255,112, 66]
* VAL     Light Purple  [255,140,255]
* THR     Dark Orange   [184, 76,  0]
* LYS     Royal Blue    [ 71, 71,184]
* ASP     Dark Rose     [160,  0, 66]
* ILE     Dark Green    [  0, 76,  0]
* ASN     Light Salmon  [255,124,112]
* GLU     Dark Brown    [102,  0,  0]
* PRO     Dark Grey     [ 82, 82, 82]
* ARG     Dark Blue     [  0,  0,124]
* PHE     Olive Grey    [ 83, 76, 66]
* GLN     Dark Salmon   [255, 76, 76]
* TYR     Medium Brown  [140,112, 76]
* HIS     Medium Blue   [112,112,255]
* CYS     Medium Yellow [255,255,112]
* MET     Light Brown   [184,160, 66]
* TRP     Olive Brown   [ 79, 70,  0]
* ASX     Medium Purple [255,  0,255]
* GLX     Medium Purple [255,  0,255]
* PCA     Medium Purple [255,  0,255]
* HYP     Medium Purple [255,  0,255]
* A       Light Blue    [160,160,255]
* C       Light Orange  [255,140, 75]
* G       Medium Salmon [255,112,112]
* T       Light Green   [160,255,160]
* Default Medium Purple [255,  0,255]
```

## 4.8 AtomColorSetScoped

```
void AtomColorSetScoped(const OESystem::OEColor &color,
                        unsigned int scope = BestScope)
void AtomColorSetScoped(unsigned int element, const OESystem::OEColor &color,
                        unsigned int scope = BestScope)
```

This function sets the color of the atoms in the specified scope to be the specified color.

## 4.9 AtomDarkColorsGet

```
bool AtomDarkColorsGet()
```

Returns whether or not the application is using the atom color palette designed for dark colored backgrounds. Otherwise, the application is using the atom color palette designed for light colored backgrounds. Both of these palettes can be edited in the preferences.

## 4.10 AtomDarkColorsSet

```
void AtomDarkColorsSet(bool dark)
```

Sets whether or not the application is using the atom color palette designed for dark colored backgrounds. Otherwise, the application is using the atom color palette designed for light colored backgrounds. Both of these palettes can be edited in the preferences.

## 4.11 AtomDefaultColorGet

```
OESystem::OEColor AtomDefaultColorGet()
OESystem::OEColor AtomDefaultColorGet(unsigned int elem)
```

Returns the current default atom color. The default atom color is applied to every atom that does not have its own specific default color assigned. Calling this function with the *elem* parameter returns the specific default color associated with the passed element number (e.g. carbon = 6).

## 4.12 AtomDefaultColorSet

```
void AtomDefaultColorSet(const OESystem::OEColor &color)
void AtomDefaultColorSet(unsigned int elem, const OESystem::OEColor &color)
```

Sets the current default atom color. The default atom color is applied to every atom that does not have its own specific default color assigned. Calling this function with the *elem* parameter sets the specific default color associated with the passed element number (e.g. carbon = 6).

## 4.13 AtomHaloRadiusGet

```
float AtomHaloRadiusGet()
```

Returns the current radius of atom selection halo. The atom selection halo is a transparent white surface with the specified radius drawn around the current selected set of atoms. If the DistanceControlsVisibility property is set (see DistanceControlsVisibilitySet/Get functions), only atoms falling within this radius will be displayed.

## 4.14 AtomHaloRadiusSet

```
float AtomHaloRadiusSet(float radius, bool notify=true)
```

Sets the current radius of atom selection halo. The atom selection halo is a transparent white surface with the specified radius drawn around the current selected set of atoms. If the DistanceControlsVisibility property is set (see DistanceControlsVisibilitySet/Get functions), only atoms falling within this radius will be displayed.

## 4.15 AtomHydrogenStyleGet

```
unsigned int AtomHydrogenStyleGet()
unsigned int AtomHydrogenStyleGet(const OEPropDB::OEKey &key)
```

Returns the current default hydrogen display style - 0 for no hydrogens, 1 for polar hydrogens only, and 2 for all hydrogens.

## 4.16 AtomHydrogenStyleSet

```
void AtomHydrogenStyleSet(unsigned int style)
```

Sets the current default hydrogen display style. Valid *style* parameter values are 0 for no hydrogens, 1 for polar hydrogens only, and 2 for all hydrogens.

## 4.17 AtomHydrogenStyleSetScoped

```
void AtomHydrogenStyleSetScoped(unsigned int style,
                                unsigned int scope = BestScope)
```

Sets the hydrogen display style for all the molecules in the specified scope. Valid *style* parameter values are 0 for no hydrogens, 1 for polar hydrogens only, and 2 for all hydrogens.

## 4.18 AtomLabelDefaultGet

```
std::string AtomLabelDefaultGet()
```

Returns the default label displayed on atoms. The default is “None”. See the AtomLabelSetScoped function for more information about label types.

## 4.19 AtomLabelDefaultSet

```
void AtomLabelDefaultSet(const std::string &label)
```

Sets the default label displayed on atoms. The default is "None". See the AtomLabelSetScoped function for more information about label types.

## 4.20 AtomLabelGet

```
std::string AtomLabelGet(const OEPropDB::OEKey &k)
```

Returns the current label set on the specified atom.

## 4.21 AtomLabelSet

```
bool AtomLabelSet(const OEPropDB::OEKey &k, const std::string &label)
```

Sets the label on the specified atom. The parameter *label* contains the text of the label and the parameter *key* specifies which atom this label should apply to. See the AtomLabelSetScoped function for more information about label types.

## 4.22 AtomLabelSetScoped

```
bool AtomLabelSetScoped(const std::string &label,
                        unsigned int scope = BestScope)
```

Sets the current label displayed on the atoms in the specified scope. Label types include:

```
* "none"               or ""
* "index"              or "%i"
* "element"            or "%e"
* "name"               or "%n"
* "type"               or "%t"
* "inttype"            or "%it"
* "implicit hcount"    or "%h"
* "formal charge"      or "%f"
* "partial charge"     or "%p"
* "residue info"       or "%r"
* "degree"             or "%d"
* "ischiral"           or "%c"
* "chirality"          or "%chi"
* "hasstereospecified" or "%sts"
* "symmetry class"     or "%sy"
* "isinring"           or "%rng"
* "oechem idx"         or "%oei"
* "map index"          or "%m"
* "read order"         or "%ro"
* "isotope"            or "%iso"
* "hcount"             or "%h"
* "residue info (ca)"  or "%car"
* "bfactor"            or "%cab"
```

```
* "degree"             or "%d"
* "hybrid"             or "%hyb"
* "comment"            or "%cm"
* "%gd(x)" - where x specifies the desired piece of generic data to use as the label
```

## 4.23 AtomStyleGet

```
std::string AtomStyleGet()
```

Returns the default atom style. The default is "small". See the AtomStyleSetScoped function for more information about atom styles.

## 4.24 AtomStyleGetScoped

```
unsigned int AtomStyleGetScoped(unsigned int scope = BestScope)
```

Returns the style of the atoms in the specified scope. If there are multiple styles present, the return type is "mixed". See the AtomStyleSetScoped function for more information about atom styles.

## 4.25 AtomStyleLargeGet

```
std::string AtomStyleLargeGet()
```

Returns the default atom style for large molecules, where large is determined by whether or not the number of atoms in the molecule exceeds the specified cutoff. For more information on this cutoff, see the MoleculeSizeCutoff function. The default is "none". See the AtomStyleSetScoped function for more information about atom styles.

## 4.26 AtomStyleLargeSet

```
void AtomStyleLargeSet(unsigned int style)
void AtomStyleLargeSet(const std::string &atomStyle)
```

Sets the default atom style for large molecules, where large is determined by whether or not the number of atoms in the molecule exceeds the specified cutoff. This function accepts either a string or an enumerated representation as its parameter. The default is "none". For more information on this large molecule atom cutoff, see the MoleculeSizeCutoff function. See the AtomStyleSetScoped function for more information about atom styles.

## 4.27 AtomStyleNucleicGet

```
std::string AtomStyleNucleicGet()
```

Returns the default atom style for nucleic acids. The default is to mirror the large molecule atom style. See the AtomStyleLargeSet function for more information about large molecules and see the AtomStyleSetScoped function for more information about atom styles.

Note: this function is a placeholder for a future implementation and therefore does not currently have any effect.

## 4.28 AtomStyleNucleicSet

```
void AtomStyleNucleicSet(unsigned int style)
void AtomStyleNucleicSet(const std::string &atomStyle)
```

Sets the default atom style for nucleic acids. This function accepts either a string or an enumerated representation as its parameter. The default is to mirror the large molecule atom style. See the AtomStyleLargeSet function for more information about large molecules and see the AtomStyleSetScoped function for more information about atom styles.

Note: this function is a placeholder for a future implementation and therefore does not currently have any effect.

## 4.29 AtomStyleProteinGet

```
std::string AtomStyleProteinGet()
```

Returns the default atom style for proteins. The default is to mirror the large molecule atom style. See the AtomStyleLargeSet function for more information about large molecules and see the AtomStyleSetScoped function for more information about atom styles.

Note: this function is a placeholder for a future implementation and therefore does not currently have any effect.

## 4.30 AtomStyleProteinSet

```
void AtomStyleProteinSet(unsigned int style)
void AtomStyleProteinSet(const std::string &atomStyle)
```

Sets the default atom style for proteins. The default is to mirror the large molecule atom style. See the AtomStyleLargeSet function for more information about large molecules and see the AtomStyleSetScoped function for more information about atom styles.

Note: this function is a placeholder for a future implementation and therefore does not currently have any effect.

## 4.31 AtomStyleSet

```
void AtomStyleSet(unsigned int style)
void AtomStyleSet(const std::string &atomStyle)
```

Sets the default atom style. This function accepts either a string or an enumerated representation as its parameter. The default is "small". See the AtomStyleSetScoped function for more information about atom styles.

## 4.32 AtomStyleSetScoped

```
void AtomStyleSetScoped(unsigned int style,
                        unsigned int scope = BestScope)
void AtomStyleSetScoped(const std::string &atomStyle,
                        unsigned int scope = BestScope)
```

Sets the display style for the specified atoms. This function accepts either a string or an enumerated representation as its parameter. Available atom styles include:

```
* "small"   atoms are drawn as small sized spheres
* "medium"  atoms are drawn as medium sized spheres
* "large"   atoms are drawn as large sized spheres
* "stars"   atoms are drawn as stars
* "radii"   atoms are drawn as spheres with radii equivalent to their VDW radii
* "none"    atoms are not drawn, but are still considered visible and thus selectable
* "hidden"  atoms are not drawn and are not considered visible and thus are not selectable
```

## 4.33 AtomStyleToEnum

```
unsigned int AtomStyleToEnum(const std::string &style)
```

This function returns the enumerated type associated the specified string atoms style representation. See the AtomStyleSetScoped function for more information about atom styles.

## 4.34 AtomStyleToText

```
std::string AtomStyleToText(unsigned int style)
```

This function returns the text string associated with the specified enumerated atom style representation. See the AtomStyleSetScoped function for more information about atom styles.

## 4.35 BondBallToStickRatioGet

```
float BondBallToStickRatioGet()
```

This function is currently a placeholder for future functionality.

## 4.36 BondBallToStickRatioSet

```
float BondBallToStickRatioSet(float ballStickRatio)
```

This function is currently a placeholder for future functionality.

## 4.37 BondClearColorScoped

```
void BondClearColorScoped(unsigned int scope = BestScope)
```

This function restores the default bond coloring to those bonds within the specified scope.

## 4.38 BondColorSetScoped

```
void BondColorSetScoped(const OESystem::OEColor &color,
                        unsigned int scope = BestScope)
```

This function sets the color of the bonds in the specified scope to be the specified color $c$.

## 4.39 BondDrawAromaticGet

```
bool BondDrawAromaticGet(unsigned int id)
```

Returns whether or not aromatic bonds should be drawn as aromatic or instead in Kekule format for the molecule with the specified *id*.

## 4.40 BondDrawAromaticSet

```
bool BondDrawAromaticSet(unsigned int id, bool aromatic)
```

Sets whether or not aromatic bonds should be drawn as aromatic or instead in kekule format for the molecule with the specified *id*.

## 4.41 BondHideHydrogenGet

```
bool BondHideHydrogenGet()
```

Returns the global property which specifies whether or not explicit hydrogens should be displayed.

## 4.42 BondHideHydrogenSet

```
bool BondHideHydrogenSet(bool hide)
```

Sets the global property which specifies whether or not explicit hydrogens should be displayed.

## 4.43 BondHideNonbondedGet

```
bool BondHideNonbondedGet()
```

Returns the global property which specifies whether or not explicit non-bonded atoms should be displayed.

## 4.44 BondHideNonbondedSet

```
bool BondHideNonbondedSet(bool hide)
```

Sets the global property which specifies whether or not explicit non-bonded atoms should be displayed.

## 4.45 BondLabelDefaultGet

```
std::string BondLabelDefaultGet()
```

Returns the default label displayed on bonds. The default is “None”. See the BondLabelSetScoped function for more information about label types.

## 4.46 BondLabelDefaultSet

```
void BondLabelDefaultSet(const std::string &label)
```

Sets the default label displayed on bonds. The default is "None". See the BondLabelSetScoped function for more information about label types.

## 4.47 BondLabelGet

```
std::string BondLabelGet(const OEPropDB::OEKey &k)
```

Returns the current label on the specified bond.

## 4.48 BondLabelSet

```
bool BondLabelSet(const OEPropDB::OEKey &k, const std::string &label)
```

Sets the label on the specified bond. The parameter *label* contains the text of the label and the parameter *key* specifies which bond this label should apply to. See the BondLabelSetScoped function for more information about label types.

## 4.49 BondLabelSetScoped

```
bool BondLabelSetScoped(const std::string &label,
                        unsigned int scope = BestScope)
```

Sets the current label displayed on the bonds in the specified scope. Label types include:

```
* "none"               or ""
* "index"              or "%i"
* "order"              or "%o"
* "length"             or "%l"
* "type"               or "%t"
* "inttype"            or "%it"
* "ischiral"           or "%c"
* "chirality"          or "%chi"
* "hasstereospecified" or "%sts"
* "oechem idx"         or "%oei"
* "%gd(x)" - where x specifies the desired piece of generic data to use as the label
```

## 4.50 BondLineWidthGet

```
float BondLineWidthGet()
```

Returns the line width (in pixels) to be used when drawing bonds in "line" or "wireframe" styles.

## 4.51 BondLineWidthSet

```
float BondLineWidthSet(float width)
bool BondLineWidthSet(const OEPropDB::OEKey &k, float width)
```

Sets the line width (in pixels) to be used when drawing bonds in “line” or “wireframe” styles.

## 4.52 BondRadiusGet

```
float BondRadiusGet()
```

Returns the radius of the cylinders to be used when drawing bonds in “cylinder” style.

## 4.53 BondRadiusSet

```
float BondRadiusSet(float radius)
bool BondRadiusSet(const OEPropDB::OEKey &k, float radius)
```

Sets the radius of the cylinders to be used when drawing bonds in “cylinder” style.

## 4.54 BondShowAromaticGet

```
bool BondShowAromaticGet()
```

Returns the global property which specifies whether or not aromatic bonds should be drawn as aromatic or instead in kekule format.

## 4.55 BondShowAromaticSet

```
bool BondShowAromaticSet(bool show)
```

Sets the global property which specifies whether or not aromatic bonds should be drawn as aromatic or instead in kekule format.

## 4.56 BondShowDipolarGet

```
bool BondShowDipolarGet()
```

Returns the global property which specifies whether or not dipolar bonds should be drawn as directed bonds.

## 4.57 BondShowDipolarSet

```
bool BondShowDipolarSet(bool show)
```

Sets the global property which specifies whether or not dipolar bonds should be drawn as directed bonds.

## 4.58 BondShowOrdersGet

```
bool BondShowOrdersGet()
```

Returns the global property which specifies whether or not bond orders should be displayed.

## 4.59 BondShowOrdersSet

```
bool BondShowOrdersSet(bool show)
```

Sets the global property which specifies whether or not bond orders should be displayed.

## 4.60 BondStyleGet

```
std::string BondStyleGet()
```

Returns the default bond style. The default is “cylinders”. See the BondStyleSetScoped function for more information about bond styles.

## 4.61 BondStyleGetScoped

```
unsigned int BondStyleGetScoped(unsigned int scope = BestScope)
```

Returns the style of the bonds in the specified scope. If there are multiple styles present, the return type is “mixed”. See the BondStyleSetScoped function for more information about bond styles.

## 4.62 BondStyleLargeGet

```
std::string BondStyleLargeGet()
```

Returns the default bond style for large molecules, where large is determined by whether or not the number of atoms in the molecule exceeds the specified cutoff. For more information on this cutoff, see the MoleculeSizeCutoff function. The default is “lines”. See the BondStyleSetScoped function for more information about bond styles.

## 4.63 BondStyleLargeSet

```
void BondStyleLargeSet(unsigned int style)
void BondStyleLargeSet(const std::string &bondstyle)
```

Sets the default bond style for large molecules, where large is determined by whether or not the number of atoms in the molecule exceeds the specified cutoff. This function accepts either a string or an enumerated representation as its parameter. The default is “lines”. For more information on this large molecule atom cutoff, see the MoleculeSizeCutoff function. See the BondStyleSetScoped function for more information about bond styles.

## 4.64 BondStyleNucleicGet

```
std::string BondStyleNucleicGet()
```

Returns the default bond style for nucleic acids. The default is to mirror the large molecule bond style. See the BondStyleLargeSet function for more information about large molecules and see the BondStyleSetScoped function for more information about bond styles.

Note: this function is a placeholder for a future implementation and therefore does not currently have any effect.

## 4.65 BondStyleNucleicSet

```
void BondStyleNucleicSet(unsigned int style)
void BondStyleNucleicSet(const std::string &bondstyle)
```

Sets the default bond style for nucleic acids. This function accepts either a string or an enumerated representation as its parameter. The default is to mirror the large molecule atom style. See the AtomStyleLargeSet function for more information about large molecules and see the AtomStyleSetScoped function for more information about atom styles.

Note: this function is a placeholder for a future implementation and therefore does not currently have any effect.

## 4.66 BondStyleProteinGet

```
std::string BondStyleProteinGet()
```

Returns the default bond style for proteins. The default is to mirror the large molecule bond style. See the BondStyleLargeSet function for more information about large molecules and see the BondStyleSetScoped function for more information about bond styles.

Note: this function is a placeholder for a future implementation and therefore does not currently have any effect.

## 4.67 BondStyleProteinSet

```
void BondStyleProteinSet(unsigned int style)
void BondStyleProteinSet(const std::string &bondstyle)
```

Sets the default bond style for proteins. The default is to mirror the large molecule bond style. See the BondStyleLargeSet function for more information about large molecules and see the BondStyleSetScoped function for more information about bond styles.

Note: this function is a placeholder for a future implementation and therefore does not currently have any effect.

## 4.68 BondStyleSet

```
void BondStyleSet(unsigned int style)
void BondStyleSet(const std::string &bondstyle)
```

Sets the default bond style. This function accepts either a string or an enumerated representation as its parameter. The default is “cylinders”. See the BondStyleSetScoped function for more information about bond styles.

## 4.69 BondStyleSetScoped

```
void BondStyleSetScoped(unsigned int style,
                        unsigned int scope = BestScope)
void BondStyleSetScoped(const std::string &bondstyle,
                        unsigned int scope = BestScope)
```

Sets the display style for the specified bonds. This function accepts either a string or an enumerated representation as its parameter. Available bonds styles include:

```
* "lines"     bonds are drawn as lines
* "cylinders" bonds are drawn as cylinders
* "none"      bonds are not drawn, but are still considered visible and thus selectable
* "hidden"    bonds are not drawn and are not considered visible and thus are not selectable
```

## 4.70 BondStyleToEnum

```
unsigned int BondStyleToEnum(const std::string &style)
```

This function returns the enumerated type associated the specified string bond style representation. See the BondStyleSetScoped function for more information about bond styles.

## 4.71 BondStyleToText

```
std::string BondStyleToText(unsigned int style)
```

This function returns the text string associated with the specified enumerated bond style representation. See the BondStyleSetScoped function for more information about bond styles.

## 4.72 BookmarkDelete

```
bool BookmarkDelete(const std::string &name)
```

Deletes the bookmark specified by *name*.

## 4.73 BookmarkExists

```
bool BookmarkExists(const std::string &name)
```

Returns whether or not the specified bookmark exists.

## 4.74 BookmarkLoad

```
bool BookmarkLoad(const std::string &name)
```

Loads the specified bookmark.

## 4.75 BookmarkMoveDown

```
bool BookmarkMoveDown(const std::string &name)
```

Moves the specified bookmark down in the order of the current set of bookmarks.

## 4.76 BookmarkMoveUp

```
bool BookmarkMoveUp(const std::string &name)
```

Moves the specified bookmark up in the order of the current set of bookmarks.

## 4.77 BookmarkOrganizeDialog

```
std::string BookmarkOrganizeDialog(bool exec=false)
```

Launches an interactive dialog which allows the user to delete, rename, and reorder the current set of bookmarks.

## 4.78 PromptBookmarkAnimationTime

```
int PromptBookmarkAnimationTime( int originalTime )
```

Launches an interactive dialog which allows the user to specify the animation duration when transitioning to a selected bookmark. Two options are provided:

```
* "Use calculated animation time"          automatically calculated using limits on speed of rotatio
* "Specify animation time in milliseconds" specified by the user.
```

## 4.79 BookmarkRename

```
bool BookmarkRename(const std::string &oldName, const std::string &newName)
```

Changes the name of the bookmark specified by *oldName* to the name specified by *newName*.

## 4.80 BookmarkSave

```
bool BookmarkSave(const std::string &name)
```

Saves the current state of the application to a named bookmark specified by the parameter *name*.

## 4.81 Bookmarks

```
std::vector<std::string> Bookmarks()
```

Returns a list of the current set of bookmarks.

## 4.82 BookmarksClear

```
void BookmarksClear()
```

Clears the current list of bookmarks.

## 4.83 BookmarksLastLoaded

```
unsigned int BookmarksLastLoaded()
```

Returns the index (in the order of the current set of bookmarks) of the last bookmark that was loaded.

## 4.84 CATraceRadiusGet

```
float CATraceRadiusGet()
```

Returns the global property which specifies the width of CA Traces.

## 4.85 CATraceRadiusSet

```
float CATraceRadiusSet(float radius)
```

Sets the global property which specifies the width of CA Traces.

## 4.86 CATraceStyleGet

```
std::string CATraceStyleGet()
std::string CATraceStyleGet(const OEPropDB::OEKey &k)
```

Returns the global property which specifies the style in which CA Traces are drawn. Available styles include: “lines” and “cylinders”.

## 4.87 CATraceStyleSet

```
std::string CATraceStyleSet(const std::string &style)
std::string CATraceStyleSet(const OEPropDB::OEKey &k, const std::string &style)
```

Sets the global property which specifies the style in which CA Traces are drawn. Available styles include: “lines” and “cylinders”.

## 4.88 CATraceStyleSetScoped

```
std::string CATraceStyleSetScoped(const std::string &str,
                                  unsigned int scope = BestScope)
```

Sets the property which specifies the style in which CA Traces are drawn. Available styles include: "lines" and "cylinders".

## 4.89 CenterSet

```
void CenterSet(const OEPropDB::OEKey &)
void CenterSet(const std::vector<OEPropDB::OEKey> &)
```

Sets the center of the scene to be at the center of the object(s) referenced by the specified key(s).

## 4.90 CenterSetScoped

```
void CenterSetScoped(unsigned int scope)
```

Scoped version of `CenterSet`.

## 4.91 ColorGet

```
OESystem::OEColor ColorGet(const OEPropDB::OEKey &)
```

Returns the color of the object specified by the OEKey parameter *key*.

## 4.92 ColorSet

```
void ColorSet(const OEPropDB::OEKey &, const OESystem::OEColor &)
```

Sets the color of the object specified by the OEKey parameter *key* to be the specified *color*.

## 4.93 ColorSetScoped

```
void ColorSetScoped(const OESystem::OEColor &,
                    unsigned int scope = BestScope)
```

Sets the color for all of the objects in the specified scope to be the specified *color*.

## 4.94 ColorUniqueScoped

```
void ColorUniqueScoped(unsigned int scope = BestScope)
```

Assigns a "unique" color to each individual object found in the specified *scope*. For large numbers of objects in the scope, the colors used may eventually repeat.

## 4.95 ContourAutoCenterGet

```
bool ContourAutoCenterGet(unsigned int id=0)
```

Returns whether or not reentrant grids are automatically contoured at the center of the scene.

## 4.96 ContourAutoCenterSet

```
bool ContourAutoCenterSet(unsigned int id, bool center)
```

Sets whether or not reentrant grids are automatically contoured at the center of the scene. If this is set to true, translating the center of the scene will cause the grids to be recontoured as the scene moves.

## 4.97 ContourAutoContourGet

```
bool ContourAutoContourGet()
```

Returns whether or not reentrant grids are automatically recontoured when the center of the scene is changed.

## 4.98 ContourAutoContourRadiusScaleSet

```
bool ContourAutoContourRadiusScaleSet(float radius, float scale)
```

Sets the radius and scale parameters used in determining how to contour a reentrant grid around the center of the scene.

## 4.99 ContourAutoContourSet

```
void ContourAutoContourSet(bool autocontour)
```

Sets whether or not reentrant grids are automatically recontoured when the center of the scene is changed.

## 4.100 ContourCenter

```
std::vector<float> ContourCenter(unsigned int id=0)
std::vector<float> ContourCenter(unsigned int id, std::vector<float>)
```

Returns the scene center position associated with the specified contour.

## 4.101 ContourColorForIndexGet

```
OESystem::OEColor ContourColorForIndexGet(unsigned int id, unsigned int index)
```

Return the color of the grid contour specified by the grid *id* and contour *index*.

## 4.102 ContourColorForIndexGetScoped

```
OESystem::OEColor ContourColorForIndexGetScoped(unsigned int index,
                                                unsigned int scope = BestScope)
```

Returns the color of the grid contours in the specified scope at contour index *index*. This color is determined by looking at the first contour found in the specified scope.

## 4.103 ContourColorForIndexSet

```
OESystem::OEColor ContourColorForIndexSet(unsigned int id, unsigned int index,
                                          const OESystem::OEColor &color)
```

For grid *id* set the color of contour *index* to *color*.

## 4.104 ContourColorForIndexSetScoped

```
OESystem::OEColor ContourColorForIndexSetScoped(unsigned int index,
                                                const OESystem::OEColor &color,
                                                unsigned int scope = BestScope)
```

For all grids in *scope* set the color of contour *index* to *color*.

## 4.105 ContourColorSet

```
void ContourColorSet(const OEPropDB::OEKey &, const OESystem::OEColor &)
```

This function sets the color of the contour associated with the specified key.

## 4.106 ContourColorSetScoped

```
void ContourColorSetScoped(const OESystem::OEColor &,
                           unsigned int scope = BestScope)
```

This function sets the color of the contours in the specified scope.

## 4.107 ContourDrawAsSurfaceGet

```
bool ContourDrawAsSurfaceGet(unsigned int id=0)
```

Returns whether or not the contour should be drawn as a solid surface or as a mesh.

## 4.108 ContourDrawAsSurfaceSet

```
bool ContourDrawAsSurfaceSet(unsigned int id, bool draw)
```

Sets whether or not the contour should be drawn as a solid surface or as a mesh.

## 4.109 ContourDrawAsSurfaceSetScoped

```
void ContourDrawAsSurfaceSetScoped(bool draw,
                                   unsigned int scope = BestScope)
```

Returns the style of contour drawing for the surfaces in scope; essentially it returns the state of the first grid found in scope.

See `ContourDrawAsSurfaceGet`.

## 4.110 ContourDrawStyleGet

```
std::string ContourDrawStyleGet(unsigned int id=0)
```

If "surface" the contours are drawn as a surface, otherwise the contour is drawn as a mesh.

## 4.111 ContourDrawStyleSet

```
void ContourDrawStyleSet(unsigned int id, const std::string &)
```

If set to "surface" the contours are drawn as a surface, otherwise the contour is drawn as a mesh.

## 4.112 ContourDrawStyleSetScoped

```
void ContourDrawStyleSetScoped(const std::string &style,
                               unsigned int scope = BestScope)
```

Set the drawstyle to all contours in scope.

## 4.113 ContourEntireGridGet

```
bool ContourEntireGridGet(unsigned int id=0)
```

Returns True if grid *id* is contoured over the whole grid. Returns False if only the portion near the center of the scene is contoured.

## 4.114 ContourEntireGridSet

```
bool ContourEntireGridSet(unsigned int id, bool entire)
```

If true contour the entire extent of grid *id*, otherwise only contour the portion near the center of the drawing scene.

## 4.115 ContourHideIndex

```
void ContourHideIndex(unsigned int index, bool hide)
void ContourHideIndex(unsigned int id, unsigned int index, bool hide)
```

This function controls the visibility of the contour specified by the grid *id* and contour *index*. If the *hide* parameter is set to true, the contour is hidden, otherwise the contour is shown.

## 4.116 ContourHideIndexGet

```
bool ContourHideIndexGet(unsigned int id, unsigned int index)
```

This function returns whether or not the contour specified by the grid *id* and contour *index* is hidden or not.

## 4.117 ContourHideIndexSet

```
void ContourHideIndexSet(unsigned int index, bool hide)
void ContourHideIndexSet(unsigned int id, unsigned int index, bool hide)
```

This function controls the visibility of the contour specified by the grid *id* and contour *index*. If the *hide* parameter is set to true, the contour is hidden, otherwise the contour is shown.

## 4.118 ContourLineWidthGet

```
float ContourLineWidthGet(unsigned int id=0)
```

Returns the line width for drawing mesh contours.

## 4.119 ContourLineWidthSet

```
float ContourLineWidthSet(unsigned int id, float rad)
```

Set the line width to *width* for drawing mesh contours.

## 4.120 ContourPickIsoSurfacesGet

```
bool ContourPickIsoSurfacesGet()
```

Returns whether or not contours are selectable by picking.

## 4.121 ContourPickIsoSurfacesSet

```
void ContourPickIsoSurfacesSet(bool pickable)
```

Sets whether or not contours are selectable by picking.

## 4.122 ContourTransparencySet

```
void ContourTransparencySet(const OEPropDB::OEKey &, unsigned int)
```

Sets the transparency of the contour associated with the specified key. The transparency is determined by the *alpha* parameter which corresponds to the color alpha value which can range from 0 (completely transparent) to 255 (completely opaque).

## 4.123 DefaultColorLabelGet

```
OESystem::OEColor DefaultColorLabelGet()
```

Returns the default color for a label.

## 4.124 DefaultColorLabelSet

```
OESystem::OEColor DefaultColorLabelSet(const OESystem::OEColor &)
```

Returns the default color for a label.

## 4.125 DefaultColorMarkedGet

```
OESystem::OEColor DefaultColorMarkedGet()
```

Returns the default color used when displaying marked objects.

## 4.126 DefaultColorMarkedSet

```
OESystem::OEColor DefaultColorMarkedSet(const OESystem::OEColor &)
```

Sets the default color used when displaying marked objects.

## 4.127 DefaultColorReferenceGet

```
OESystem::OEColor DefaultColorReferenceGet()
```

Returns the default reference molecule color.

## 4.128 DefaultColorReferenceSet

```
OESystem::OEColor DefaultColorReferenceSet(const OESystem::OEColor &)
```

Sets the default reference molecule color.

## 4.129 DefaultColorSelectedGet

```
OESystem::OEColor DefaultColorSelectedGet()
```

Returns the default selection color.

## 4.130 DefaultColorSelectedSet

```
OESystem::OEColor DefaultColorSelectedSet(const OESystem::OEColor &)
```

Sets the default selection color.

## 4.131 DefaultColorTitleGet

```
OESystem::OEColor DefaultColorTitleGet()
```

Returns the default title color.

## 4.132 DefaultColorTitleSet

```
OESystem::OEColor DefaultColorTitleSet(const OESystem::OEColor &)
```

Sets the default title color.

## 4.133 DefaultMonitorColorGet

```
OESystem::OEColor DefaultMonitorColorGet()
```

Returns the default monitor color.

## 4.134 DefaultMonitorColorSet

```
OESystem::OEColor DefaultMonitorColorSet(const OESystem::OEColor &)
```

Sets the default monitor color.

## 4.135 DistanceControlsVisibilityGet

```
bool DistanceControlsVisibilityGet()
```

Returns whether or not an atom's or bond's distance from the selected set will affect its visibility. The distance cutoff is specified by the AtomHaloSet function.

## 4.136 DistanceControlsVisibilitySet

```
void DistanceControlsVisibilitySet(bool state)
```

Sets whether or not an atom's or bond's distance from the selected set will affect its visibility. The distance cutoff is specified by the AtomHaloSet function.

## 4.137 DrawAtomsAndBondsGet

```
bool DrawAtomsAndBondsGet()
```

Returns whether or not the application has drawing of atoms and bonds enabled. Ordinarily, this function should not need to be called by the user.

## 4.138 DrawAtomsAndBondsSet

```
bool DrawAtomsAndBondsSet(bool draw)
```

Sets whether or not the application has drawing of atoms and bonds enabled. Ordinarily, this function should not need to be called by the user.

## 4.139 DrawAxesGet

```
bool DrawAxesGet()
```

Returns whether or not axes should be drawn in the application's 3D display.

## 4.140 DrawAxesSet

```
bool DrawAxesSet(bool draw)
```

Sets whether or not axes should be drawn in the application's 3D display.

## 4.141 DrawCATracesGet

```
bool DrawCATracesGet()
bool DrawCATracesGet(const OEPropDB::OEKey &k)
```

Returns whether or not the application has drawing of CA Traces enabled. Ordinarily, this function should not need to be called by the user.

## 4.142 DrawCATracesSet

```
bool DrawCATracesSet(bool draw)
bool DrawCATracesSet(const OEPropDB::OEKey &, bool draw)
```

Sets whether or not the application has drawing of CA Traces enabled. Ordinarily, this function should not need to be called by the user.

## 4.143 DrawCATracesSetScoped

```
bool DrawCATracesSetScoped(bool, unsigned int scope = BestScope)
```

Sets whether or not a CA Trace should be drawn the proteins in the specified scope.

## 4.144 DrawContoursGet

```
bool DrawContoursGet()
```

Returns true if the contours are being drawn. Returns false if no contours are being drawn.

## 4.145 DrawContoursSet

```
bool DrawContoursSet(bool draw)
```

If *draw* is true, then draw all contours for each visible grid. Otherwise no contours are drawn.

## 4.146 DrawLabelsGet

```
bool DrawLabelsGet()
```

Returns whether or not the application has drawing of labels enabled.

## 4.147 DrawLabelsSet

```
bool DrawLabelsSet(bool draw)
```

Sets whether or not the application has drawing of labels enabled.

## 4.148 DrawMatrixGet

```
bool DrawMatrixGet()
```

Returns whether or not the application has matrix (or tiled) displays enabled.

## 4.149 DrawMatrixSet

```
bool DrawMatrixSet(bool draw)
```

Sets whether or not the application has matrix (or tiled) displays enabled.

## 4.150 DrawRibbonsGet

```
bool DrawRibbonsGet()
bool DrawRibbonsGet(const OEPropDB::OEKey &k)
```

Returns whether or not the application has drawing of protein ribbons enabled. Ordinarily, this function should not need to be called by the user.

## 4.151 DrawRibbonsSet

```
bool DrawRibbonsSet(bool draw)
bool DrawRibbonsSet(const OEPropDB::OEKey &k, bool draw)
```

Sets whether or not the application has drawing of protein ribbons enabled. Ordinarily, this function should not need to be called by the user.

## 4.152 DrawRibbonsSetScoped

```
bool DrawRibbonsSetScoped(bool draw,
                          unsigned int scope = BestScope)
```

Sets whether or not a ribbon display should be shown for the proteins in the specified scope.

## 4.153 DrawSurfacesGet

```
bool DrawSurfacesGet()
```

Returns true if surfaces are being drawn, false if not.

## 4.154 DrawSurfacesSet

```
bool DrawSurfacesSet(bool draw)
```

If parameter *draw* if false, surface objects are not drawn. If *draw* is true surfaces are drawn.

## 4.155 DrawSymmetryGet

```
bool DrawSymmetryGet()
```

Returns whether or not the application has drawing of symmetry enabled.

## 4.156 DrawSymmetrySet

```
bool DrawSymmetrySet(bool draw)
```

Sets whether or not the application has drawing of symmetry enabled.

## 4.157 DrawTitlesGet

```
bool DrawTitlesGet()
```

Returns whether or not the application has drawing of titles enabled.

## 4.158 DrawTitlesSet

```
bool DrawTitlesSet(bool draw)
```

Sets whether or not the application has drawing of symmetry enabled.

## 4.159 DrawUnitCellGet

```
bool DrawUnitCellGet()
```

Returns whether or not the application has drawing of unit cells enabled.

## 4.160 DrawUnitCellSet

```
bool DrawUnitCellSet(bool draw)
```

Sets whether or not the application has drawing of symmetry enabled.

## 4.161 GridDefaultContourColorByIndexGet

```
OESystem::OEColor GridDefaultContourColorByIndexGet(unsigned int gridType,
                                                    unsigned int index)
```

Returns the default contour color for contour *index* for grids of type *gridType*.

## 4.162 GridDefaultContourColorByIndexSet

```
void GridDefaultContourColorByIndexSet(unsigned int gridType,
                                       unsigned int index,
                                       const OESystem::OEColor &color)
```

Set the default contour *color* for contour *index* for grids of type *gridType*.

## 4.163 GridDefaultDrawAsSurfaceGet

```
bool GridDefaultDrawAsSurfaceGet(unsigned int gridType)
```

Returns the default drawing style for contours of grid with type *gridType*.

If true, then the surface is drawn as a solid surface, otherwise the surface is drawn as a mesh.

## 4.164 GridDefaultDrawAsSurfaceSet

```
void GridDefaultDrawAsSurfaceSet(unsigned int gridType, bool asSurf)
```

Set the default drawing style *asSurf* for contours of grid with type *gridType*.

If *asSurf* is true, then the surface is drawn as a solid surface, otherwise the surface is drawn as a mesh.

## 4.165 GridShowCornersGet

```
bool GridShowCornersGet()
```

Returns true if grid corners are being shown false if not.

Grid corners are indicators of the extent of visible grids. They are useful for displaying uncontoured grids and showing whether grid space and molecules intersect.

## 4.166 GridShowCornersSet

```
void GridShowCornersSet(bool show)
```

Set *show* to true to show grid corners, false if not.

Grid corners are indicators of the extent of visible grids. They are useful for displaying uncontoured grids and showing whether grid space and molecules intersect.

## 4.167 GridShowLastMaskedGrid

```
unsigned int GridShowLastMaskedGrid(bool show=true)
```

Reserved for internal use.

## 4.168 HBondAddTarget

```
void HBondAddTarget(const OEPropDB::OEKey &key)
```

This function adds the molecule corresponding to the specified *key* to the list of targets to be considered when displaying hydrogen bonds. Calling this function on a molecule will cause hydrogens to be added to that molecule if they are not already present.

## 4.169 HBondAddTargetsScoped

```
void HBondAddTargetsScoped(unsigned int scope = BestScope)
```

This function adds of the molecules in the specified *scope* to the list of targets to be considered when displaying hydrogen bonds. Calling this function on a molecule will cause hydrogens to be added to that molecule if they are not already present.

## 4.170 HBondClearTargets

```
void HBondClearTargets()
```

This function clears the list of molecule targets to be considered when displaying hydrogen bonds.

## 4.171 HBondColorGet

```
OESystem::OEColor HBondColorGet()
```

Returns the current color being used when drawing hydrogen bonds.

## 4.172 HBondColorSet

```
void HBondColorSet(const OESystem::OEColor &)
```

Sets the color to be used when drawing hydrogen bonds.

## 4.173 HBondRemoveTarget

```
void HBondRemoveTarget(const OEPropDB::OEKey &key)
```

This function removes the molecule corresponding to the specified *key* from the list of molecule targets to be considered when displaying hydrogen bonds.

## 4.174 HBondRemoveTargetsScoped

```
void HBondRemoveTargetsScoped(unsigned int scope = BestScope)
```

This function removes all of the molecules in the specified *scope* from the list of molecule targets to be considered when displaying hydrogen bonds.

## 4.175 HBondShowExternalGet

```
bool HBondShowExternalGet()
```

Returns whether or not external hydrogen bonds (those between two separate molecules) should be displayed if present.

## 4.176 HBondShowExternalSet

```
void HBondShowExternalSet(bool show)
```

Sets whether or not external hydrogen bonds (those between two separate molecules) should be displayed if present.

## 4.177 HBondShowInternalGet

```
bool HBondShowInternalGet()
```

Returns whether or not internal hydrogen bonds should be displayed if present.

## 4.178 HBondShowInternalSet

```
void HBondShowInternalSet(bool show)
```

Sets whether or not internal hydrogen bonds should be displayed if present.

## 4.179 HaloColorDefaultGet

```
OESystem::OEColor HaloColorDefaultGet()
```

Returns the default color to be used when displaying halos.

## 4.180 HaloColorDefaultSet

```
void HaloColorDefaultSet(const OESystem::OEColor &c)
```

Sets the default color to be used when displaying halos.

## 4.181 HaloColorGet

```
OESystem::OEColor HaloColorGet(const OEPropDB::OEKey &)
```

Returns the color of the halo associated with the specified atom key.

## 4.182 HaloColorSet

```
void HaloColorSet(const OEPropDB::OEKey &k, const std::string &)
void HaloColorSet(const OEPropDB::OEKey &k, const OESystem::OEColor &c)
```

Sets the color of the halo associated with the specified atom key.

## 4.183 HaloColorSetScoped

```
void HaloColorSetScoped(const std::string &,
                        unsigned int scope = BestScope)
void HaloColorSetScoped(const OESystem::OEColor &,
                        unsigned int scope = BestScope)
```

Sets the color of the halos associated with the atoms in the specified scope.

## 4.184 HaloRadiusGet

```
float HaloRadiusGet(const OEPropDB::OEKey &)
```

Returns the radius of the halo associated with the specified atom key.

## 4.185 HaloRadiusSet

```
void HaloRadiusSet(const OEPropDB::OEKey &, float)
void HaloRadiusSet(const OEPropDB::OEKey &, const std::string &)
```

Sets the radius of the halo associated with the specified atom key.

## 4.186 HaloRadiusSetScoped

```
void HaloRadiusSetScoped(float, unsigned int scope = BestScope)
void HaloRadiusSetScoped(const std::string &,
                         unsigned int scope = BestScope)
```

Sets the radius of the halos associated with the atoms in the specified scope.

## 4.187 HaloScaleGet

```
float HaloScaleGet(const OEPropDB::OEKey &)
```

Returns the scaling factor for the size of the halo associated with the specified atom key.

## 4.188 HaloScaleSet

```
void HaloScaleSet(const OEPropDB::OEKey &, float)
void HaloScaleSet(const OEPropDB::OEKey &, const std::string &)
```

Sets the scaling factor for the size of the halo associated with the specified atom key.

## 4.189 HaloScaleSetScoped

```
void HaloScaleSetScoped(float, unsigned int scope = BestScope)
void HaloScaleSetScoped(const std::string &,
                        unsigned int scope = BestScope)
```

Sets the scaling factor for the size of the halo associated with the atoms in the specified scope.

## 4.190 HideNoneScoped

```
void HideNoneScoped(unsigned int scope = BestScope)
```

Makes visible any hidden atoms or bonds on all the molecules in the specified scope.

## 4.191 HideOthers

```
void HideOthers(const std::vector<OEPropDB::OEKey> &keys, bool value)
```

Hides everything except the objects specified by list of keys passed as a parameter to this function.

## 4.192 HideScoped

```
void HideScoped(bool value, unsigned int scope = BestScope)
```

Hides everything in the specified scope.

## 4.193 LabelClearColorScoped

```
void LabelClearColorScoped(unsigned int scope = BestScope)
```

This function restores the default label coloring to those labels within the specified scope.

## 4.194 LabelClearScoped

```
void LabelClearScoped(unsigned int scope = BestScope)
```

Clears the labels on all the atoms and bonds in the specified scope.

## 4.195 LabelColorScoped

```
void LabelColorScoped(const OESystem::OEColor &c,
                      unsigned int scope = BestScope)
```

This function sets the color of the labels in the specified scope to be the specified color *c*.

## 4.196 LabelDefaultColorGet

```
OESystem::OEColor LabelDefaultColorGet()
```

This function returns the default color for labels.

## 4.197 LabelDefaultColorSet

```
void LabelDefaultColorSet(const OESystem::OEColor &c)
```

This function sets the default color for labels.

## 4.198 LabelFixedSizeGet

```
bool LabelFixedSizeGet()
```

Returns whether or not labels are drawn using a fixed size on the screen or whether they scale with zooming of the scene.

## 4.199 LabelFixedSizeSet

```
void LabelFixedSizeSet(bool fixed)
```

Sets whether or not labels are drawn using a fixed size on the screen or whether they scale with zooming of the scene.

## 4.200 LabelGet

```
std::string LabelGet(const OEPropDB::OEKey &key, bool full=false,
                     bool coords=true)
```

Returns a description of the specified object. If the *full* parameter is True, it will also include information from the parent object (if one exists). The *coords* parameter specifies whether or not coordinates should be included in the description (primarily for atoms and vertices).

## 4.201 MoleculeAltLocationShow

```
bool MoleculeAltLocationShow(const OEPropDB::OEKey &, unsigned int, bool)
bool MoleculeAltLocationShow(const OEPropDB::OEKey &, unsigned int, unsigned int,
                             unsigned int)
```

Sets the visibility of the specified alternate locations for the specified molecule. This function is primarily intended for internal use only.

## 4.202 MoleculeAltLocationVisible

```
bool MoleculeAltLocationVisible(const OEPropDB::OEKey &, unsigned int)
bool MoleculeAltLocationVisible(const OEPropDB::OEKey &, unsigned int,
                                unsigned int)
```

Returns whether or the specified alternate locations on the specified molecule are visible. This function is primarily intended for internal use only.

## 4.203 MoleculeAtomBondStyleSetScoped

```
void MoleculeAtomBondStyleSetScoped(const std::string &atomStyle,
                                    const std::string &bondStyle,
                                    unsigned int scope = BestScope)
```

Set the combined style for atoms and bonds.

- atomStyle is the style to set the atom , see AtomStyleSetScoped.
- bondStyle is the style to set the bond, see BondStyleSetScoped.
- scope is the default scope to use.

## 4.204 MoleculeColorByScoped

```
void MoleculeColorByScoped(const std::string &property,
                           const std::string &params="",
                           unsigned int scope = BestScope)
```

This color colors all the molecules in the specified scope according to the scheme specified by the *property* parameter. Valid properties include:

* "amino"
* "bfactor"
* "chain"
* "charge"
* "cpk"
* "cpknew"
* "formal charge"
* "group"
* "hbond"
* "hydrophobicity"
* "partial charge"
* "shapely"

If the specified property is *hydrophobicity*, the specific scale to be used can be specified in the *params* parameter. Valid scale options include:

* "charifson"
* "eisenberg"
* "kytedolittle"
* "whiteoctanol"

For more details on the specifics of these coloring scheme, please see the user manual.

## 4.205 MoleculeColorsResetScoped

```
void MoleculeColorsResetScoped(unsigned int scope = BestScope)
```

This function restores the default atom and bond coloring to those atoms and bonds in the specified scope. It is essentially equivalent to calling AtomClearColorScoped followed by BondClearColorScoped.

## 4.206 MoleculeDarkColorsGet

```
bool MoleculeDarkColorsGet()
```

Returns whether or not the application is using either the atom or residue color palette designed for dark colored backgrounds. Otherwise, the application is using the atom color palettes designed for light colored backgrounds. All of these palettes can be edited in the preferences. For more information, see the functions AtomDarkColorsGet and ResidueDarkColorsGet.

## 4.207 MoleculeDarkColorsSet

```
void MoleculeDarkColorsSet(bool dark)
```

Sets whether or not the application is using the atom and residue color palettes designed for dark colored backgrounds. Otherwise, the application is using the atom color palettes designed for light colored backgrounds. All of these palettes can be edited in the preferences. For more information, see the functions AtomDarkColorsSet and ResidueDarkColorsSet.

## 4.208 MoleculeShowfAntsyGet

```
bool MoleculeShowfAntsyGet()
```

Returns whether or not the application is using the high quality molecule representations as opposed to standard models.

## 4.209 MoleculeShowfAntsySet

```
bool MoleculeShowfAntsySet(bool show)
```

Sets whether or not the application is using the high quality molecule representations as opposed to standard models.

## 4.210 MoleculeStyleGet

```
std::string MoleculeStyleGet()
```

Returns the default molecule style. The default is "stick". See the MoleculeStyleSetScoped function for more information about molecule styles.

## 4.211 MoleculeStyleGetScoped

```
unsigned int
  MoleculeStyleGetScoped(unsigned int scope = BestScope)
```

Returns the style of the molecules in the specified scope. If there are multiple styles present, the return type is "mixed". See the MoleculeStyleSetScoped function for more information about molecule styles.

## 4.212 MoleculeStyleLargeGet

```
std::string MoleculeStyleLargeGet()
```

Returns the default molecule style for large molecules, where large is determined by whether or not the number of atoms in the molecule exceeds the specified cutoff. For more information on this cutoff, see the MoleculeSizeCutoff function. The default is "wireframe". See the MoleculeStyleSetScoped function for more information about molecule styles.

## 4.213 MoleculeStyleLargeSet

```
void MoleculeStyleLargeSet(unsigned int style)
void MoleculeStyleLargeSet(const std::string &style)
```

Sets the default molecule style for large molecules, where large is determined by whether or not the number of atoms in the molecule exceeds the specified cutoff. For more information on this cutoff, see the MoleculeSizeCutoff function. The default is "wireframe". See the MoleculeStyleSetScoped function for more information about molecule styles.

## 4.214 MoleculeStyleNucleicGet

```
std::string MoleculeStyleNucleicGet()
```

Returns the default molecule style for nucleic acids. The default is to mirror the large molecule style. See the MoleculeStyleLargeSet function for more information about large molecules and see the MoleculeStyleSetScoped function for more information about molecule styles.

Note: this function is a placeholder for a future implementation and therefore does not currently have any effect.

## 4.215 MoleculeStyleNucleicSet

```
void MoleculeStyleNucleicSet(unsigned int style)
void MoleculeStyleNucleicSet(const std::string &style)
```

Sets the default molecule style for nucleic acids. The default is to mirror the large molecule style. See the MoleculeStyleLargeSet function for more information about large molecules and see the MoleculeStyleSetScoped function for more information about molecule styles.

Note: this function is a placeholder for a future implementation and therefore does not currently have any effect.

## 4.216 MoleculeStyleProteinGet

```
std::string MoleculeStyleProteinGet()
```

Returns the default molecule style for proteins. The default is to mirror the large molecule style. See the MoleculeStyleLargeSet function for more information about large molecules and see the MoleculeStyleSetScoped function for more information about molecule styles.

Note: this function is a placeholder for a future implementation and therefore does not currently have any effect.

## 4.217 MoleculeStyleProteinSet

```
void MoleculeStyleProteinSet(unsigned int style)
void MoleculeStyleProteinSet(const std::string &style)
```

Sets the default molecule style for proteins. The default is to mirror the large molecule style. See the MoleculeStyleLargeSet function for more information about large molecules and see the MoleculeStyleSetScoped function for more information about molecule styles.

Note: this function is a placeholder for a future implementation and therefore does not currently have any effect.

## 4.218 MoleculeStyleSet

```
void MoleculeStyleSet(unsigned int style)
void MoleculeStyleSet(const std::string &atomStyle)
```

Sets the default molecule style. This function accepts either a string or an enumerated representation as its parameter. The default is "stick". See the MoleculeStyleSetScoped function for more information about molecule styles.

## 4.219 MoleculeStyleSetScoped

```
void MoleculeStyleSetScoped(unsigned int style,
                            unsigned int scope = BestScope)
void MoleculeStyleSetScoped(const std::string &atomStyle,
                            unsigned int scope = BestScope)
```

Sets the display style for the specified molecules. This function accepts either a string or an enumerated representation as its parameter. Available molecule styles include:

```
* "ball and stick" molecules are drawn in ball and stick mode
* "cpk"            molecules are drawn in CPK mode
* "stars"          molecules are drawn in star mode
* "stick"          molecules are drawn in stick mode
* "wireframe"      molecules are drawn in wireframe mode
* "none"           molecules are not drawn, but are still considered visible and thus selectable
* "hidden"         molecules are not drawn and are not considered visible and thus are not selectable
```

## 4.220 MoleculeStyleToEnum

```
unsigned int MoleculeStyleToEnum(const std::string &style)
```

This function returns the enumerated type associated the specified string molecule style representation. See the MoleculeStyleSetScoped function for more information about molecule styles.

## 4.221 MoleculeStyleToText

```
std::string MoleculeStyleToText(unsigned int style)
```

This function returns the text string associated with the specified enumerated molecule style representation. See the MolStyleSetScoped function for more information about molecule styles.

## 4.222 MonitorAngleCreate

```
unsigned int MonitorAngleCreate(const OEPropDB::OEKey &key1,
                                const OEPropDB::OEKey &key2,
                                const OEPropDB::OEKey &key3)
```

Attempts to create an angle monitor between the atoms specified by the parameters *key1*, *key2*, and *key3*.

Returns the ID of the created monitor or 0 if one could not be created.

## 4.223 MonitorAngleDelete

```
void MonitorAngleDelete(const OEPropDB::OEKey &key1,
                        const OEPropDB::OEKey &key2,
                        const OEPropDB::OEKey &key3)
```

Deletes the monitor, if any, specified by atom keys: *key1*, *key2*, *key3*.

## 4.224 MonitorAngleExists

```
unsigned int MonitorAngleExists(const OEPropDB::OEKey &key1,
                                const OEPropDB::OEKey &key2,
                                const OEPropDB::OEKey &key3)
```

Returns the ID of the angle monitor defined by the specified atom keys or zero if no such monitor exists.

## 4.225 MonitorColorGet

```
OESystem::OEColor MonitorColorGet(const OEPropDB::OEKey &monitor)
```

Returns the color of the specified monitor.

## 4.226 MonitorColorSet

```
void MonitorColorSet(const OEPropDB::OEKey &monitor,
                     const OESystem::OEColor &clr)
```

Sets the color of the specified monitor.

## 4.227 MonitorDeleteScoped

```
void MonitorDeleteScoped(unsigned int scope = BestScope)
```

Deletes all the monitors in the specified scope.

## 4.228 MonitorDistanceCreate

```
unsigned int MonitorDistanceCreate(const OEPropDB::OEKey &key1,
                                   const OEPropDB::OEKey &key2)
```

Attempts to create a distance monitor between the two atoms specified by the parameters *key1* and *key2*.

Returns the ID of the created monitor or 0 if one could not be created.

## 4.229 MonitorDistanceDelete

```
void MonitorDistanceDelete(const OEPropDB::OEKey &key1,
                           const OEPropDB::OEKey &key2)
```

Deletes the monitor specified by the two atom keys.

## 4.230 MonitorDistanceExists

```
unsigned int MonitorDistanceExists(const OEPropDB::OEKey &key1,
                                   const OEPropDB::OEKey &key2)
```

Returns the ID of the distance monitor determined by the two atom keys or zero if no such monitor exists.

## 4.231 MonitorSphereCreate

```
unsigned int MonitorSphereCreate(const OEPropDB::OEKey &key1,
                                 const std::string &name,
                                 const OESystem::OEColor &clr, float radius=0)
unsigned int MonitorSphereCreate(const OEPropDB::OEKey &parent,
                                 const std::string &name, float x, float y,
                                 float z, float radius,
                                 const OESystem::OEColor &color)
```

Create a sphere monitor on an atom specified by *key1* or at the coordinates specified by *x*, *y*, and *z*. The monitor is only visible when the specified atom key or parent key (in the case of coordinate specification) is visible.

Returns the ID of the newly created monitor or 0 if one could not be created.

## 4.232 MonitorTorsionCreate

```
unsigned int MonitorTorsionCreate(const OEPropDB::OEKey &key1,
                                  const OEPropDB::OEKey &key2,
                                  const OEPropDB::OEKey &key3,
                                  const OEPropDB::OEKey &key4)
```

Attempts to create a torsion monitor between the atoms specified by *key1*, *key2*, *key3*, and *key4*.

Returns the ID of the created monitor or 0 if one could not be created.

## 4.233 MonitorTorsionDelete

```
void MonitorTorsionDelete(const OEPropDB::OEKey &key1,
                          const OEPropDB::OEKey &key2,
                          const OEPropDB::OEKey &key3,
                          const OEPropDB::OEKey &key4)
```

Deletes the torsion monitor, if any, specified by the atom keys: *key1*, *key2*, *key3*, *key4*.

## 4.234 MonitorTorsionExists

```
unsigned int MonitorTorsionExists(const OEPropDB::OEKey &key1,
                                  const OEPropDB::OEKey &key2,
                                  const OEPropDB::OEKey &key3,
                                  const OEPropDB::OEKey &key4)
```

Returns the ID of the torsion monitor determined by the specified four atom keys or zero if no such monitor exists.

## 4.235 MonitorsVisible

```
bool MonitorsVisible()
```

This function returns whether any monitors are currently visible.

## 4.236 MonitorsVisibleDelete

```
void MonitorsVisibleDelete()
```

Deletes all monitors that are currently being displayed.

## 4.237 PaneActivated

```
void PaneActivated(unsigned int pane)
```

In matrix (tiled) mode, sets the specified pane to be the active pane in the display. Ordinarily, this function should not need to be called by the user.

## 4.238 ProteinColorByBFactor

```
void ProteinColorByBFactor(const OEPropDB::OEKey &key)
```

Colors a protein referenced by key. Atoms with the lowest temperature factors will be colored bright blue, while those with the highest will be colored bright red.

## 4.239 ProteinColorByBFactorScoped

```
void ProteinColorByBFactorScoped(unsigned int scope, int bins=10)
void ProteinColorByBFactorScoped(unsigned int scope,
                                 const std::vector<BFactorColor> &clrs)
void ProteinColorByBFactorScoped(unsigned int scope, float min, float max,
                                 const OESystem::OEColor &clr)
```

Scoped versions of `ProteinColorByBFactor`.

## 4.240 ResidueColorPaletteUpdate

```
void ResidueColorPaletteUpdate()
```

This function updates the default residue coloring palette based on whether or not the application is using a dark or light colored background scheme. This function is called internally and should not by called by the user.

## 4.241 ResidueDarkColorsGet

```
bool ResidueDarkColorsGet()
```

Returns whether or not the application is using the residue color palette designed for dark colored backgrounds. Otherwise, the application is using the residue color palette designed for light colored backgrounds. Both of these palettes can be edited in the preferences.

## 4.242 ResidueDarkColorsSet

```
void ResidueDarkColorsSet(bool dark)
```

Sets whether or not the application is using the residue color palette designed for dark colored backgrounds. Otherwise, the application is using the residue color palette designed for light colored backgrounds. Both of these palettes can be edited in the preferences.

## 4.243 ResidueDefaultColorGet

```
OESystem::OEColor ResidueDefaultColorGet()
OESystem::OEColor ResidueDefaultColorGet(unsigned int res)
```

Returns the current default residue color. The default residue color is applied to every residue that does not have its own specific default color assigned. Calling this function with the *res* parameter returns the specific default color associated with the passed residue index.

## 4.244 ResidueDefaultColorSet

```
void ResidueDefaultColorSet(const OESystem::OEColor &c)
void ResidueDefaultColorSet(unsigned int res, const OESystem::OEColor &c)
```

Sets the current default residue color. The default residue color is applied to every residue that does not have its own specific default color assigned. Calling this function with the *res* parameter sets the specific default color associated with the passed residue index.

## 4.245 RibbonClearColorScoped

```
void RibbonClearColorScoped(unsigned int scope = BestScope)
```

This function restores the default ribbon coloring to those ribbons within the specified *scope*.

## 4.246 RibbonColorSetScoped

```
void RibbonColorSetScoped(const OESystem::OEColor &c,
                          unsigned int scope = BestScope)
```

This function sets the color of the ribbons in the specified *scope* to be the specified color *c*.

## 4.247 RibbonCrossResolutionGet

```
unsigned int RibbonCrossResolutionGet()
```

Returns the cross resolution used in calculating protein ribbons.

## 4.248 RibbonCrossResolutionSet

```
unsigned int RibbonCrossResolutionSet(unsigned int res)
```

Sets the cross resolution used in calculating protein ribbons.

## 4.249 RibbonGapGet

```
float RibbonGapGet()
```

Returns the gap value used in calculating protein ribbons.

## 4.250 RibbonGapSet

```
float RibbonGapSet(float gap)
```

Sets the gap value used in calculating protein ribbons.

## 4.251 RibbonHeightScaleGet

```
float RibbonHeightScaleGet()
```

Returns the height scale used in calculating protein ribbons.

## 4.252 RibbonHeightScaleSet

```
float RibbonHeightScaleSet(float scale)
```

Sets the height scale used in calculating protein ribbons.

## 4.253 RibbonRadiusGet

```
float RibbonRadiusGet()
```

Returns the radius used in calculating protein ribbons.

## 4.254 RibbonRadiusSet

```
float RibbonRadiusSet(float radius)
```

Sets the radius used in calculating protein ribbons.

## 4.255 RibbonResolutionGet

```
unsigned int RibbonResolutionGet()
```

Returns the resolution used in calculating protein ribbons.

## 4.256 RibbonResolutionSet

```
unsigned int RibbonResolutionSet(unsigned int res)
```

Sets the resolution used in calculating protein ribbons.

## 4.257 RibbonSplineTypeGet

```
std::string RibbonSplineTypeGet()
```

Returns the spline type used in calculating protein ribbons. Available types include: “CubicHermite” and “Beta”.

## 4.258 RibbonSplineTypeSet

```
std::string RibbonSplineTypeSet(const std::string &type)
```

Sets the spline type used in calculating protein ribbons. Available types include: “CubicHermite” and “Beta”.

## 4.259 RibbonStyleGet

```
std::string RibbonStyleGet()
```

Returns the style used in displaying protein ribbons. Available types include: “Round”, “Lines”, and “Cartoon”. The default is “Cartoon”.

## 4.260 RibbonStyleSet

```
std::string RibbonStyleSet(const std::string &style)
std::string RibbonStyleSet(const OEPropDB::OEKey &k, const std::string &style)
```

Sets the style used in displaying protein ribbons. Available types include: “Round”, “Lines”, and “Cartoon”. The default is “Cartoon”.

## 4.261 RibbonStyleSetScoped

```
std::string RibbonStyleSetScoped(const std::string &style,
                                 unsigned int scope = BestScope)
```

Sets the style used in displaying ribbons for the proteins in the specified scope. Available types include: “Round”, “Lines”, and “Cartoon”. The default is “Cartoon”.

## 4.262 RibbonWidthScaleGet

```
float RibbonWidthScaleGet()
```

Returns the width scale used in calculating protein ribbons.

## 4.263 RibbonWidthScaleSet

```
float RibbonWidthScaleSet(float scale)
```

Sets the width scale used in calculating protein ribbons.

## 4.264 SceneDrawActiveBorderGet

```
bool SceneDrawActiveBorderGet()
```

Returns whether or not a border is drawn (in matrix mode) around the cell containing the focused object.

## 4.265 SceneDrawActiveBorderSet

```
void SceneDrawActiveBorderSet(bool draw)
```

Sets whether or not a border is drawn (in matrix mode) around the cell containing the focused object.

## 4.266 SceneMatrixModeGet

```
std::string SceneMatrixModeGet()
```

Returns the display style for the individual cells when the application is in matrix mode. Available styles are: “OnePerView”, “LockedOnAll”, and “LockedPerView”.

## 4.267 SceneMatrixModeSet

```
std::string SceneMatrixModeSet(std::string mode)
```

Sets the display style for the individual cells when the application is in matrix mode. Available styles are: “OnePerView”, “LockedOnAll”, and “LockedPerView”.

## 4.268 SelectionColorBlendFactorGet

```
float SelectionColorBlendFactorGet()
```

Returns the alpha buffer coloring blending factor for selection colors. The default value is 1.0 which corresponds to zero blending with the selection color being the one displayed. A value of 0.0 also corresponds to zero blending but with the underlying color being the one displayed. A value of 0.5 corresponds to even blending of the underlying color and the selected color.

## 4.269 SelectionColorBlendFactorSet

```
float SelectionColorBlendFactorSet(float alpha)
```

Sets the alpha buffer coloring blending factor for selection colors. The default value is 1.0 which corresponds to zero blending with the selection color being the one displayed. A value of 0.0 also corresponds to zero blending but with the underlying color being the one displayed. A value of 0.5 corresponds to even blending of the underlying color and the selected color.

## 4.270 ShowESGridScoped

```
void ShowESGridScoped(bool show=true,
                      unsigned int scope = BestScope)
```

If *show* is true, attach and show electrostatic grids to all molecules in *scope*.

If *show* if false, hide all attached electrostatic grids to molecules in *scope*.

## 4.271 ShowSurface

```
void ShowSurface(const OEPropDB::OEKey &, const std::string &, bool show)
```

Sets whether or not a property surface of the specified *type* is displayed for the molecule associated with the specified key. Valid types are “molecular” and “accessible”.

## 4.272 ShowSurfaceScoped

```
void ShowSurfaceScoped(const std::string &type, bool show,
                       unsigned int scope = BestScope)
```

Scoped version of `ShowSurface`.

## 4.273 SurfaceAlterTransparency

```
void SurfaceAlterTransparency(unsigned int id, unsigned int alpha)
```

Sets the transparency of the surface associated with the specified *id*. The transparency is specified by the *alpha* parameter which determines the alpha component of the surface color. The alpha values ranges from 0 (completely transparent) to 255 (completely opaque).

## 4.274 SurfaceColorBy

```
std::string SurfaceColorBy(const std::string &mode="")
void SurfaceColorBy(const OEPropDB::OEKey &key, const std::string &mode,
                    const std::string &params="")
```

This function colors the surface referenced by the specified *key* using the specified *style*. Valid styles include:

- **atom - colors the surface based on the color of the nearest atom to the** vertex being colored
- concavity - colors the surface by concavity, red is high concavity.
- curvature - colors the surface by curvature, green is positive curvature.
- **distance - colors the surface by its distance to the currently selected atoms.** Each color band represents two angstroms.
- **electrostatics - colors the surface using a red-to-blue scheme based on the** electrostatic potential observed at the surface.
- **hydrogen bonds - colors the surface by the underlying atom hydrogen-bond** donor/acceptor properties. Donors are red and acceptors are blue.
- **hydrophobicity - colors the surface using a hydrophobic color scale. There** are four different hydrophobic scales available: charifson, eisenberg, kytedolittle, and whiteoctanol. The desired scale can be set in the application preferences.
- **potential - colors the surface based using a red-to-blue scheme (low to high)** over the values in the surface's potential array. This is not the same as coloring by electrostatics.

## 4.275 SurfaceColorByScoped

```
void SurfaceColorByScoped(const std::string &colorby,
                          const std::string &params="",
                          unsigned int scope = BestScope)
```

Scoped version of `SurfaceColorBy`.

## 4.276 SurfaceColorGet

```
OESystem::OEColor SurfaceColorGet(unsigned int id)
```

Returns the color of the specified surface.

## 4.277 SurfaceColorGetScoped

```
OESystem::OEColor
  SurfaceColorGetScoped(unsigned int scope = BestScope)
```

Returns the color of the first surface found in the specified scope.

## 4.278 SurfaceColorSet

```
OESystem::OEColor SurfaceColorSet(unsigned int id,
                                  const OESystem::OEColor &color)
```

Set the color of the surface *id* to *color*.

## 4.279 SurfaceColorSetScoped

```
void SurfaceColorSetScoped(const OESystem::OEColor &color,
                           unsigned int scope = BestScope)
```

Set the color of the surfaces in *scope* to *color*.

## 4.280 SurfaceGrowTriangle

```
void SurfaceGrowTriangle(unsigned int oerid, const OEPropDB::OEKey &trikey,
                         bool addtriangle=false)
```

Grow the surface selection when a triangle is double clicked. Ordinarily this function does not need to be called.

## 4.281 SurfaceLineWidthGet

```
float SurfaceLineWidthGet(unsigned int id=0)
```

Returns the line width used when drawing the surface specified by *id* in mesh mode.

## 4.282 SurfaceLineWidthSet

```
float SurfaceLineWidthSet(unsigned int id, float width)
```

Sets the line width used when drawing the surface specified by *id* in mesh mode.

## 4.283 SurfaceStyleGet

```
std::string SurfaceStyleGet(unsigned int id)
```

Returns the default style of surfaces. The style is one of:

- mesh - the surfaces are drawn as meshes.
- points - the surfaces are drawn as points.
- solid - the surfaces are drawn as solid objects.

## 4.284 SurfaceStyleGetScoped

```
std::string SurfaceStyleGetScoped(unsigned int scope = BestScope)
```

Return the style for the surfaces in scope. The value returned is from the first surface found in scope and is not indicative of all surfaces.

See `SurfaceStyleGet` for a list of surface styles.

## 4.285 SurfaceStyleSet

```
std::string SurfaceStyleSet(unsigned int id, const std::string &style)
```

Set the surface *id* to on of the following styles:

- mesh - the surface is drawn as a mesh.
- points - the surface is drawn as points.
- solid - the surface is drawn as a solid object.

## 4.286 SurfaceStyleSetScoped

```
void SurfaceStyleSetScoped(const std::string &style,
                           unsigned int scope = BestScope)
```

Scoped version of `SurfaceStyleSet`.

## 4.287 SurfaceTransparencySet

```
void SurfaceTransparencySet(unsigned int id, unsigned int alpha)
```

Set the transparency to all surfaces in *scope* to transparency.

## 4.288 SurfaceTransparencySetScoped

```
void SurfaceTransparencySetScoped(unsigned int transparency,
                                  unsigned int scope = BestScope)
```

Set the transparency to all surfaces in *scope* to transparency.

## 4.289 SurfaceVertexFloodScoped

```
int SurfaceVertexFloodScoped(unsigned int scope = BestScope)
```

Get the surface vertex flood of all surfaces in scope. This is essentially the flood of the focused surface or the surface of the focused or first visible molecule.

## 4.290 SymmetryColorModeGet

```
unsigned int SymmetryColorModeGet()
```

## 4.291 SymmetryColorModeSet

```
unsigned int SymmetryColorModeSet(unsigned int mode)
```

## 4.292 TitleDefaultColorGet

```
OESystem::OEColor TitleDefaultColorGet()
```

Returns the default color for titles.

## 4.293 TitleDefaultColorSet

```
void TitleDefaultColorSet(const OESystem::OEColor &c)
```

Sets the default color for titles.

## 4.294 TitlesDrawAboveGet

```
bool TitlesDrawAboveGet()
```

Returns whether or not titles are drawn at the top of the scene or the bottom of the scene.

## 4.295 TitlesDrawAboveSet

```
void TitlesDrawAboveSet(bool top)
```

Sets whether or not titles are drawn at the top of the scene or the bottom of the scene.

## 4.296 TransparencySet

```
void TransparencySet(const OEPropDB::OEKey &, unsigned int alpha)
```

This function sets the transparency of the object associated with the specified key. The transparency is determined by the specified parameter *alpha* which corresponds to the desired alpha value of the object's color. Alpha can range from 0 (completely transparent) to 255 (completely opaque).

## 4.297 TransparencySetScoped

```
void TransparencySetScoped(unsigned int alpha,
                           unsigned int scope = BestScope)
```

This function sets the transparency of all the objects in the specified scope. The transparency is determined by the specified parameter *alpha* which corresponds to the desired alpha value of the object's color. Alpha can range from 0 (completely transparent) to 255 (completely opaque).

## 4.298 ViewerAmbientLightGet

```
OESystem::OEColor ViewerAmbientLightGet()
```

Returns the OpenGL ambient light property.

## 4.299 ViewerAmbientLightSet

```
OESystem::OEColor ViewerAmbientLightSet(const OESystem::OEColor &color)
```

Sets the OpenGL ambient light property.

## 4.300 ViewerAmbientMaterialGet

```
OESystem::OEColor ViewerAmbientMaterialGet()
```

Returns the OpenGL ambient material property.

## 4.301 ViewerAmbientMaterialSet

```
OESystem::OEColor ViewerAmbientMaterialSet(const OESystem::OEColor &color)
```

Sets the OpenGL ambient material property.

## 4.302 ViewerAnimate

```
void ViewerAnimate(const std::vector<float> &centers,
                   const std::vector<float> &look,
                   const std::vector<float> &up,
                   unsigned int              msec,
                   unsigned int              fps)
```

Reserved for future use.

## 4.303 ViewerAnimateTo

```
void ViewerAnimateTo( const std::string &bookmark,
                      unsigned int       msec,
                      unsigned int       fps)
void ViewerAnimateTo( const OEPropDB::OEKey &centerKey,
                      const OEPropDB::OEKey &refKey,
                      unsigned int           msec,
                      unsigned int           fps,
                      float                  zoom = 0.0 )
void ViewerAnimateTo( const std::vector<float> &center,
                      const std::vector<float> &look,
                      const std::vector<float> &up,
                      unsigned int              msec,
                      unsigned int              fps,
                      float                     zoom = 0.0 )
```

Animates the scene from the current position to the specified position. All implementations of this function take an *msec* and *fps* parameters. The *msec* parameter specifies how long it should take the animation to be performed. Please note that this time is only a suggestion, as many factors including CPU speed, GPU speed, and available memory will influence the actual performance of the animation. The *fps* parameters specifies how many frames per second should be used in the animation.

The first implementation takes a bookmark name as its only other parameter. The second implementation takes two OEKey parameters corresponding to the desired final center and viewing point positions for the scene. The third implementation takes three arrays corresponding to the center coordinates, a vector defining the viewing angle, and a vector defining the up direction for the scene.

## 4.304 ViewerAntialiasGet

```
bool ViewerAntialiasGet()
```

Returns whether or not antialiasing is turned on for line drawing.

## 4.305 ViewerAntialiasSet

```
bool ViewerAntialiasSet(bool state)
```

Sets whether or not antialiasing is turned on for line drawing.

## 4.306 ViewerAutoCenterGet

```
bool ViewerAutoCenterGet()
```

Returns whether or not the scene will automatically be recentered when a change in the scene occurs (e.g. a new object is made active, objects are hidden or shown).

## 4.307 ViewerAutoCenterPanesGet

```
bool ViewerAutoCenterPanesGet()
```

Returns whether or not the scene in each pane in multi-pane display mode is automatically centered or whether all panes observe the same center.

## 4.308 ViewerAutoCenterPanesSet

```
void ViewerAutoCenterPanesSet(bool center)
```

Sets whether or not the scene in each pane in multi-pane display mode is automatically centered or whether all panes observe the same center.

## 4.309 ViewerAutoCenterSet

```
bool ViewerAutoCenterSet(bool ac)
```

Sets whether or not the scene will automatically be recentered when a change in the scene occurs (e.g. a new object is made active, objects are hidden or shown).

## 4.310 ViewerAutoFitGet

```
bool ViewerAutoFitGet()
```

Returns whether or not the contents of the scene will automatically be fit to the screen when a scene change occurs.

## 4.311 ViewerAutoFitSet

```
bool ViewerAutoFitSet(bool af)
```

Sets whether or not the contents of the scene will automatically be fit to the screen when a scene change occurs.

## 4.312 ViewerBackgroundColorGet

```
OESystem::OEColor ViewerBackgroundColorGet()
```

Returns the background color of the 3D display.

## 4.313 ViewerBackgroundColorSet

```
OESystem::OEColor ViewerBackgroundColorSet(const OESystem::OEColor &color)
```

Sets the background color of the 3D display.

## 4.314 ViewerBookmarkLoad

```
bool ViewerBookmarkLoad(std::string bm)
```

Loads the specified bookmark.

## 4.315 ViewerBookmarksGetAnimated

```
bool ViewerBookmarksGetAnimated()
```

Returns whether or not the transition between bookmarks is animated.

## 4.316 ViewerBookmarksGetAnimationTime

```
unsigned int ViewerBookmarksGetAnimationTime()
```

Returns the default desired time for how long the transition between bookmarks should take if animated.

## 4.317 ViewerBookmarksSetAnimated

```
void ViewerBookmarksSetAnimated(bool animated)
```

Sets whether or not the transition between bookmarks is animated.

## 4.318 ViewerBookmarksSetAnimationTime

```
void ViewerBookmarksSetAnimationTime(unsigned int msecs)
```

Sets the default desired time for how long the transition between bookmarks should take if animated.

## 4.319 ViewerCenterAndRadiusGet

```
std::vector<float> ViewerCenterAndRadiusGet()
```

Returns the current center and radius of the scene in the 3D display in a four-membered list. The first three values in the list are the x,y,z coordinates of the center. The fourth value is the radius.

## 4.320 ViewerCenterAndRadiusSet

```
std::vector<float> ViewerCenterAndRadiusSet(float x, float y, float z,
                                            float radius, bool redraw=true)
```

Sets the current center and radius of the scene in the 3D display. The *redraw* parameter determines whether or not the scene should be redrawn immediately.

## 4.321 ViewerCenterGet

```
std::vector<float> ViewerCenterGet()
```

Returns the current center of the scene as a list of three values corresponding to the x,y,z coordinates. If for some reason the 3D display is invalid, the list returned will be empty.

## 4.322 ViewerCenterSet

```
std::vector<float> ViewerCenterSet(unsigned int, bool redraw=true)
std::vector<float> ViewerCenterSet(float x, float y, float z, bool redraw=true)
```

Sets the current center of the scene. The function that takes three floating point parameters explicitly sets the center of the scene to the coordinates specified by the *x*, *y*, and *z* parameters. The *redraw* parameter specifies whether or not the scene should be immediately redrawn.

The function that takes a single unsigned integer parameter (*id*) attempts to center the scene based on the geometry of the object corresponding to the *id* parameter. If for some reason, a center could not be calculated, this function will return an empty list, otherwise it will return the new center.

## 4.323 ViewerCenterSetScoped

```
std::vector<float>
  ViewerCenterSetScoped(unsigned int scope = BestScope,
                        bool redraw=true)
```

Sets the current center of the scene based on the geometry of the objects found within the specified scope. The *redraw* parameter specifies whether or not the scene should be immediately redrawn. If for some reason, a center could not be calculated, this function will return an empty list, otherwise it will return the new center.

## 4.324 ViewerDepthCueFollowsSlab

```
bool ViewerDepthCueFollowsSlab()
bool ViewerDepthCueFollowsSlab(bool enable)
```

Sets whether or not the depthcue parameters should be adjusted as the slabbing parameters change to maintain the best possible lighting for the scene.

## 4.325 ViewerDepthcueEndGet

```
float ViewerDepthcueEndGet()
```

Returns the spatial end point used in depthcue calculations. Ordinarily, this function should not need to be called by the user.

## 4.326 ViewerDepthcueEndSet

```
float ViewerDepthcueEndSet(float end, bool redraw=true)
```

Sets the spatial end point used in depthcue calculations. Ordinarily, this function should not need to be called by the user.

## 4.327 ViewerDepthcueGet

```
bool ViewerDepthcueGet()
```

Returns whether or not depthcueing is currently enabled in the 3D display.

## 4.328 ViewerDepthcueSet

```
bool ViewerDepthcueSet(bool state)
```

Sets whether or not depthcueing is currently enabled in the 3D display.

## 4.329 ViewerDepthcueStartGet

```
float ViewerDepthcueStartGet()
```

Returns the spatial start point used in depthcue calculations. Ordinarily, this function should not need to be called by the user.

## 4.330 ViewerDepthcueStartSet

```
float ViewerDepthcueStartSet(float start, bool redraw=true)
```

Sets the spatial start point used in depthcue calculations. Ordinarily, this function should not need to be called by the user.

## 4.331 ViewerDiffuseLightGet

```
OESystem::OEColor ViewerDiffuseLightGet()
```

Returns the OpenGL diffuse light property.

## 4.332 ViewerDiffuseLightSet

```
OESystem::OEColor ViewerDiffuseLightSet(const OESystem::OEColor &color)
```

Sets the OpenGL diffuse light property.

## 4.333 ViewerDiffuseMaterialGet

```
OESystem::OEColor ViewerDiffuseMaterialGet()
```

Returns the OpenGL diffuse material property.

## 4.334 ViewerDiffuseMaterialSet

```
OESystem::OEColor ViewerDiffuseMaterialSet(const OESystem::OEColor &color)
```

Set the OpenGL diffuse material property.

## 4.335 ViewerDrawDepictionsGet

```
bool ViewerDrawDepictionsGet()
```

Returns whether or not a 2D depiction is drawn in the 3D display for the active molecule.

## 4.336 ViewerDrawDepictionsSet

```
bool ViewerDrawDepictionsSet(bool show)
```

Sets whether or not a 2D depiction is drawn in the 3D display for the active molecule.

## 4.337 ViewerFit

```
void ViewerFit(float buffer=0.0f)
void ViewerFit(const std::vector<OEPropDB::OEKey> &keys, float buffer=0.0f)
```

This function ensures that the entire contents of the scene fit within the 3D display window. The *buffer* parameter allows specification of an additional buffer applied to the radius of the scene.

The function that takes a list of keys as a parameter will perform the same function as above, but will only ensure that those objects corresponding to the specified keys will be guaranteed to fit within the 3D display window.

## 4.338 ViewerFontSizeGet

```
unsigned int ViewerFontSizeGet()
```

Return the size of the font used for text drawn in the 3D display.

## 4.339 ViewerFontSizeSet

```
void ViewerFontSizeSet(unsigned int sz)
```

Sets the size of the font used for text drawn in the 3D display.

## 4.340 ViewerForwardGet

```
std::vector<float> ViewerForwardGet()
```

Returns the forward facing vector for the OpenGL camera.

## 4.341 ViewerLODGet

```
int ViewerLODGet()
```

Returns the level of display (rendering quality) being used in 3D display window.

## 4.342 ViewerLODSet

```
int ViewerLODSet(unsigned int lod, bool redraw=true)
```

Sets the level of display (rendering quality) being used in the 3D display.

## 4.343 ViewerLightPositionGet

```
std::vector<float> ViewerLightPositionGet()
```

Returns the position of the OpenGL light source.

## 4.344 ViewerLightPositionSet

```
std::vector<float> ViewerLightPositionSet(const std::vector<float> &pos)
```

Sets the position of the OpenGL light source.

## 4.345 ViewerLookAt

```
void ViewerLookAt(const OEPropDB::OEKey &centerKey,
                  const OEPropDB::OEKey &refKey, bool redraw=true)
void ViewerLookAt(const std::vector<float> &center,
                  const std::vector<float> &look,
                  const std::vector<float> &up, bool redraw=true)
```

Sets up the display such that the object corresponding to *centerKey* is at the center of the display with the camera looking down the axis between *centerKey* and the object corresponding to *refKey*.

## 4.346 ViewerMirrorSlabsGet

```
bool ViewerMirrorSlabsGet()
```

Returns whether or not the position of the front and back clipping planes mirror each other.

## 4.347 ViewerMirrorSlabsSet

```
bool ViewerMirrorSlabsSet(bool mirror)
```

Sets whether or not the position of the front and back clipping planes mirror each other.

## 4.348 ViewerNiceFontsGet

```
bool ViewerNiceFontsGet()
```

Returns whether or not text is rendered in the 3D display using high-quality fonts versus simple fonts. Some graphics cards cannot properly handle the high-quality fonts and it is recommended in those situations not to use "nice" fonts.

## 4.349 ViewerNiceFontsSet

```
bool ViewerNiceFontsSet(bool nice)
```

Sets whether or not text is rendered in the 3D display using high-quality fonts versus simple fonts. Some graphics cards cannot properly handle the high-quality fonts and it is recommended in those situations not to use "nice" fonts.

## 4.350 ViewerOrientationGet

```
std::vector<float> ViewerOrientationGet()
```

Returns the OpenGL camera orientation matrix.

## 4.351 ViewerOrientationSet

```
std::vector<float> ViewerOrientationSet(std::vector<float> orientationMatrix)
```

Sets the OpenGL camera orientation matrix.

## 4.352 ViewerProjectorModeGet

```
bool ViewerProjectorModeGet()
```

Returns whether or not the display is being drawn in "Projector Mode". "Projector Mode" is a display mode which is designed to improve visibility when the application is projected using an LCD projector by using a lighter background and a different atom coloring scheme.

## 4.353 ViewerProjectorModeSet

```
void ViewerProjectorModeSet(bool state)
```

Sets whether or not the display is being drawn in "Projector Mode". "Projector Mode" is a display mode which is designed to improve visibility when the application is projected using an LCD projector by using a lighter background and a different atom coloring scheme.

## 4.354 ViewerRadiusGet

```
float ViewerRadiusGet()
```

Returns the current radius of the scene in the 3D display window.

## 4.355 ViewerRadiusSet

```
float ViewerRadiusSet(float radius, bool redraw=true)
```

Sets the current radius of the scene in the 3D display window. The *redraw* parameter specifies whether or not to immediately redraw the scene.

## 4.356 ViewerRecenter

```
void ViewerRecenter()
```

This function recenters the scene based on the geometry of the contents of the scene.

## 4.357 ViewerReprobeStereo

```
bool ViewerReprobeStereo()
```

This function rechecks to see whether hardware stereo is supported on the current machine. Ordinarily, this function should not need to be called by the user.

## 4.358 ViewerRotate

```
void ViewerRotate(const std::string &axis, float angle, bool redraw=true)
```

This function rotates the display around the specified axis (valid parameters: 'x','y','z' for screen or trackball axis, 'X', 'Y', 'Z' for world axis) by *angle* degrees. The center of rotation is the world center. The *redraw* parameter specifies whether or not to immediately redraw the scene.

## 4.359 ViewerScaleGet

```
float ViewerScaleGet()
```

Returns the scale of the scene in the 3D display window.

## 4.360 ViewerScaleSet

```
float ViewerScaleSet(float radius, bool redraw=true)
```

Sets the scale of the scene in the 3D display window.

## 4.361 ViewerShininessMaterialGet

```
float ViewerShininessMaterialGet()
```

Returns the OpenGL material shininess property.

## 4.362 ViewerShininessMaterialSet

```
float ViewerShininessMaterialSet(float shine)
```

Sets the OpenGL material shininess property.

## 4.363 ViewerShowActiveBorderGet

```
bool ViewerShowActiveBorderGet()
```

Returns whether or not a blue border is drawn around the active pane in multi-pane display mode.

## 4.364 ViewerShowActiveBorderSet

```
void ViewerShowActiveBorderSet(bool show)
```

Sets whether or not a blue border is drawn around the active pane in multi-pane display mode.

## 4.365 ViewerShowGridGet

```
bool ViewerShowGridGet()
```

Returns whether or not borders are drawn around individual panes in multi-pane display mode.

## 4.366 ViewerShowGridSet

```
void ViewerShowGridSet(bool show)
```

Sets whether or not borders are drawn around individual panes in multi-pane display mode.

## 4.367 ViewerShowTrackballGuideSet

```
void ViewerShowTrackballGuideSet(bool show)
```

Sets whether or not a trackball guide will be displayed on the 3D display window to help guide what is considered inside and outside with respect to mouse actions. Please see ViewerMouseOutsideAwareGet/Set functions for more details.

## 4.368 ViewerSlabEnableGet

```
bool ViewerSlabEnableGet()
```

Returns whether or not slabbing (clipping) of the 3D display is enabled.

## 4.369 ViewerSlabEnableSet

```
bool ViewerSlabEnableSet(bool enable)
```

Sets whether or not slabbing (clipping) of the 3D display is enabled. The *redraw* parameter specifies whether or not the scene should be immediately redrawn.

## 4.370 ViewerSlabFarGet

```
float ViewerSlabFarGet()
```

Returns the spatial position of the far clipping plane. Ordinarily, this function should not need to be called by the user.

## 4.371 ViewerSlabFarSet

```
float ViewerSlabFarSet(float radius, bool redraw=true)
```

Sets the spatial position of the far clipping plane. The *redraw* parameter specifies whether or not the scene should be immediately redrawn. Ordinarily, this function should not need to be called by the user.

## 4.372 ViewerSlabNearGet

```
float ViewerSlabNearGet()
```

Returns the spatial position of the near clipping plane. Ordinarily, this function should not need to be called by the user.

## 4.373 ViewerSlabNearSet

```
float ViewerSlabNearSet(float radius, bool redraw=true)
```

Sets the spatial position of the near clipping plane. The *redraw* parameter specifies whether or not the scene should be immediately redrawn. Ordinarily, this function should not need to be called by the user.

## 4.374 ViewerSlabWidthGet

```
float ViewerSlabWidthGet()
```

Returns the distance between the near and far clipping planes. Ordinarily, this function should not need to be called by the user.

## 4.375 ViewerSlabWidthSet

```
float ViewerSlabWidthSet(float width, bool redraw=true)
```

Sets the distance between the near and far clipping planes. The *redraw* parameter specifies whether or not the scene should be immediately redrawn. Ordinarily, this function should not need to be called by the user.

## 4.376 ViewerSpecularMaterialGet

```
OESystem::OEColor ViewerSpecularMaterialGet()
```

Returns the OpenGL specular material property.

## 4.377 ViewerSpecularMaterialSet

```
OESystem::OEColor ViewerSpecularMaterialSet(const OESystem::OEColor &color)
```

Sets the OpenGL specular material property.

## 4.378 ViewerStereoAngleGet

```
float ViewerStereoAngleGet()
```

Returns the angle used when calculating stereographic display offsets.

## 4.379 ViewerStereoAngleSet

```
float ViewerStereoAngleSet(float angle)
```

Sets the angle used when calculating stereographic display offsets.

## 4.380 ViewerStereoCrossEyedGet

```
bool ViewerStereoCrossEyedGet()
```

Returns whether or not the stereographic display is in cross-eyed mode.

## 4.381 ViewerStereoCrossEyedSet

```
bool ViewerStereoCrossEyedSet(bool cross)
```

Sets whether or not the stereographic display is in cross-eyed mode.

## 4.382 ViewerStereoEnableGet

```
bool ViewerStereoEnableGet()
```

Returns whether or not stereographic display is enabled.

## 4.383 ViewerStereoEnableSet

```
bool ViewerStereoEnableSet(bool enabled)
```

Sets whether or not stereographic display is enabled.

## 4.384 ViewerStereoHardwareGet

```
bool ViewerStereoHardwareGet()
```

Returns whether or not stereo hardware is being used for the stereographic display.

## 4.385 ViewerStereoHardwareSet

```
bool ViewerStereoHardwareSet(bool hardware)
```

Sets whether or not stereo hardware is being used for the stereographic display.

## 4.386 ViewerStereoSeparationGet

```
float ViewerStereoSeparationGet()
```

Returns the eye separation used in stereographic display mode.

## 4.387 ViewerStereoSeparationSet

```
float ViewerStereoSeparationSet(float separation)
```

Sets the eye separation used in stereographic display mode.

## 4.388 ViewerStereoStyleGet

```
unsigned char ViewerStereoStyleGet()
```

Sets the stereographic display mode (hardware, splitscreen, or none).

## 4.389 ViewerStereoStyleSet

```
unsigned char ViewerStereoStyleSet(unsigned char style)
```

Sets the stereographic display mode (hardware, stencil, splitscreen, or none).

## 4.390 ViewerStyleControlVisibleGet

```
bool ViewerStyleControlVisibleGet(const std::string &style)
```

Returns whether or not the specified style control widget is currently visible above the 3D display. Available style widgets include: “color”, “selection”, “style”, “contours”, and “graphics”.

## 4.391 ViewerStyleControlVisibleSet

```
void ViewerStyleControlVisibleSet(const std::string &style, bool vis)
```

Sets whether or not the specified style control widget is currently visible above the 3D display. Available style widgets include: “color”, “selection”, “style”, “contours”, and “graphics”.

## 4.392 ViewerSupportsHWStereo

```
bool ViewerSupportsHWStereo()
```

Returns whether or not the machine the application is being run on supports the use hardware stereo.

## 4.393 ViewerTextFontGet

```
std::string ViewerTextFontGet()
```

Returns the font currently being used when rendering text in the 3D display. This font only applies when using "nice" fonts.

## 4.394 ViewerTextFontSet

```
void ViewerTextFontSet(const std::string &)
```

Sets the font currently being used when rendering text in the 3D display. This font only applies when using "nice" fonts.

## 4.395 ViewerTextScaleGet

```
float ViewerTextScaleGet()
```

Returns the current global scale for text drawn in the 3D display window.

## 4.396 ViewerTextScaleSet

```
float ViewerTextScaleSet(float scale, bool redraw=true)
```

Sets the current global scale for text drawn in the 3D display window.

## 4.397 ViewerToggleRenderFeatures

```
void ViewerToggleRenderFeatures(unsigned int features, bool on)
```

Toggles the use of certain advanced visualization features.

## 4.398 ViewerTranslateX

```
void ViewerTranslateX(float val)
```

Translates the display along the X axis by the specified amount (in Angstroms).

## 4.399 ViewerTranslateY

```
void ViewerTranslateY(float val)
```

Translates the display along the Y axis by the specified amount (in Angstroms).

## 4.400 ViewerTranslateZ

```
void ViewerTranslateZ(float val)
```

Translates the display along the Z axis by the specified amount (in Angstroms).

## 4.401 ViewerUpGet

```
std::vector<float> ViewerUpGet()
```

Returns the OpenGL camera up vector.

## 4.402 ViewerUseDisplayListGet

```
bool ViewerUseDisplayListGet()
```

Returns whether or not the 3D renderer uses a display list to encapsulate the master scene. The default is false. Ordinarily, this function does not need to be called by the user.

## 4.403 ViewerUseDisplayListSet

```
bool ViewerUseDisplayListSet(bool use)
```

Sets whether or not the 3D renderer uses a display list to encapsulate the master scene. The default is false. Ordinarily, this function does not need to be called by the user. Setting this property to true may have cause significant negative performance on certain machines.

## 4.404 ViewerUseSystemFontsGet

```
bool ViewerUseSystemFontsGet()
```

Returns whether or not the 3D renderer uses system fonts when displaying text.

## 4.405 ViewerUseSystemFontsSet

```
bool ViewerUseSystemFontsSet(bool state)
```

Sets whether or not the 3D renderer uses system fonts when display text.

## 4.406 ViewerSetShowObjectToolbar

```
void ViewerSetShowObjectToolbar(bool state)
```

Sets whether or not the style & color buttons are shown in the bottom toolbar for the active object

## 4.407 ViewerGetShowObjectToolbar

```
bool ViewerGetShowObjectToolbar()
```

Sets whether or not the style & color buttons are shown in the bottom toolbar for the active object

CHAPTER

FIVE

# OBJECT FUNCTIONS

The functions detailed in this section provide a mechanism to add, delete, modify, and organize the many types of objects (e.g. molecules, grids, and surfaces) available in VIDA.

## 5.1 Add

```
unsigned int Add(OESystem::OEBase &, unsigned int listID = 0)
```

Adds the specified object to the application and returns a unique ID associated with that object. If the optional listID argument is provided, then the object is added to the specified list. Otherwise, a new list is created and the object is added to it.

## 5.2 AddCSVSmiles

```
unsigned int AddCSVSmiles( const std::string &smiles,
                           const std::vector<std::string> &headers,
                           const std::vector<std::string> &values,
                           const std::string &filename = "",
                           unsigned int       listid   = 0)
```

Adds a new molecule from SMILES with the optional associated data. *smiles* species the SMILES string to parse, *headers* is a list of column names corresponding to the data specified in *values*, *filename* is an optional filename to attribute this molecule to, and *listid* is the ID of the list to which this molecule will be added. If *listid* is zero, a new list will be created.

Returns the ID of the list to which the molecule was added.

## 5.3 AddURLMol

```
unsigned int AddURLMol( const std::string              &url,
                        const std::string              &urlFunc,
                        const std::vector<std::string> &headers,
                        const std::vector<std::string> &values,
                        const std::string              &filename,
                        unsigned int                    listid)
```

Reserved for future implementation.

## 5.4 AtomDataGet

```
AtomData AtomDataGet(const OEPropDB::OEKey &key )
```

Returns an AtomData object populated with information about the atom specified by the parameter *key*.

## 5.5 CheckIn

```
bool CheckIn(OESystem::OEBase &)
```

Checks the specified Python accessible object back into the main VIDA application which will synchronize any changes made to the object in Python with the object in VIDA. The object in question must have been checked out of VIDA prior to being checked back in.

See `GridCheckOut`, `MoleculeCheckOut`, and `SurfaceCheckOut`.

## 5.6 ChildrenGet

```
std::vector<OEPropDB::OEKey> ChildrenGet(unsigned int id)
std::vector<OEPropDB::OEKey> ChildrenGet(const OEPropDB::OEKey &)
```

Returns a list of keys corresponding to any child objects associated with the specified object.

## 5.7 ContourCloudJitterDefaultGet

```
float ContourCloudJitterDefaultGet()
```

Returns the value (between 0.0 and 1.0) indicated by the Jitter slider on the Contours panel.

## 5.8 ContourCloudJitterGet

```
float ContourCloudJitterGet(unsigned int id)
```

Returns the jitter value assigned to grid *id*. The jitter value is used when the contour is displayed with the "cloud" style, and represents the degree to which the grid vertex positions are to be randomized. Jitter values range from 0.0 (no jitter) to 1.0 (jitter equal to the grid spacing).

## 5.9 ContourCloudJitterSet

```
float ContourCloudJitterSet(unsigned int id, float jitter)
```

Sets the jitter value to be used when grid *id* is displayed with the "cloud" style. Jitter values range from 0.0 (no jitter) to 1.0 (jitter equal to the grid spacing).

## 5.10 ContourCountScoped

```
int ContourCountScoped(unsigned int scope = BestScope)
```

Returns the number of contours for all grids in *scope*.

This returns the number of contours for the focused grid or the first grid found in the given scope.

## 5.11 ContourCreate

```
void ContourCreate(unsigned int id=0)
void ContourCreate(unsigned int id, float threshold)
void ContourCreate(unsigned int id, float threshold,
                   const OESystem::OEColor &color)
```

Create a new contour for grid *id*.

## 5.12 ContourCreateSurface

```
unsigned int ContourCreateSurface(unsigned int id, unsigned int contourIndex)
```

Create a contour surface for the surface defined by the repository id *id* for the contour referenced by *contourIndex*.

## 5.13 ContourDeleteAll

```
void ContourDeleteAll(unsigned int id)
```

Delete all contours for grid *id*.

## 5.14 ContourDeleteByIndex

```
void ContourDeleteByIndex(unsigned int id, unsigned int index)
```

Delete contour *index* for grid *id*. All contours with indices greater than *index* will now be at their current index - 1.

## 5.15 ContourDeleteByThreshold

```
void ContourDeleteByThreshold(unsigned int id, float threshold)
```

Delete all contours above *threshold* for grid *id*. Contour indices will be reassigned.

## 5.16 ContourExtractIsoSurface

```
unsigned int ContourExtractIsoSurface(unsigned int id, unsigned int vertex,
                                      bool maskVis=true)
```

Reserved for internal use.

## 5.17 ContourFixAsSurfaceScoped

```
std::vector<unsigned int>
  ContourFixAsSurfaceScoped(unsigned int scope = BestScope)
```

Creates an actual surface object for each contour in the specified scope. Returns a list of IDs corresponding to the generated surfaces.

## 5.18 ContourGetAll

```
std::vector<float> ContourGetAll(unsigned int id=0)
```

Get all contour levels from each contour drawn for grid *id*.

## 5.19 ContourLevelForIndexGet

```
float ContourLevelForIndexGet(unsigned int id, unsigned int index)
```

Returns the contour level for grid *id* at index *id*.

## 5.20 ContourLevelForIndexGetScoped

```
float ContourLevelForIndexGetScoped(int index,
                                    unsigned int scope = BestScope)
```

Returns the contour level for index *id* for the first grid in *scope*.

## 5.21 ContourLevelForIndexSet

```
float ContourLevelForIndexSet(unsigned int id, unsigned int index, float thresh)
```

Set the level for the contour at index *index* to *thresh* for grid *id*.

## 5.22 ContourLevelForIndexSetScoped

```
void ContourLevelForIndexSetScoped(unsigned int index, float thresh,
                                   unsigned int gridType=UINT_MAX,
                                   unsigned int scope = BestScope)
```

Set the level for the contour at index *index* to *thresh* for all grids in *scope*.

## 5.23 ContourLevelNudgeScoped

```
void ContourLevelNudgeScoped(bool increase,
                             unsigned int scope = BestScope)
```

This function increases or decreases the contour value for the contours in the specified scope by 0.1 depending on whether the *increase* parameter is set to true or false.

## 5.24 ContourLevelSetScoped

```
void ContourLevelSetScoped(float level,
                           unsigned int scope = BestScope)
```

This function sets the contour level of all the contours in the specified scope to the value specified by the parameter *level*.

## 5.25 ContourMax

```
float ContourMax(unsigned int oerid)
```

Returns the maximum allowable contour threshold for grid *id*.

## 5.26 ContourMaxScoped

```
float ContourMaxScoped(unsigned int scope = BestScope)
```

Returns the maximum allowable contour threshold for the first grid found in *scope*.

## 5.27 ContourMin

```
float ContourMin(unsigned int oerid)
```

Returns the minimum allowable contour threshold for grid *id*.

## 5.28 ContourMinScoped

```
float ContourMinScoped(unsigned int scope = BestScope)
```

Returns the minimum allowable contour threshold for the first grid found in *scope*.

## 5.29 ContourRadiusGet

```
float ContourRadiusGet(unsigned int id=0)
```

Returns the radius used when generating contours for crystallographic grids.

## 5.30 ContourRadiusSet

```
float ContourRadiusSet(unsigned int id, float rad)
```

Sets the radius used when generating contours for crystallographic grids.

## 5.31 ContourResolutionGet

```
float ContourResolutionGet(unsigned int id=0)
```

Return the contouring resolution for drawing the contours in Angstroms. A lower resolution samples more grid points, but requires more memory.

## 5.32 ContourResolutionSet

```
float ContourResolutionSet(unsigned int id, float rad)
```

Set the contouring resolution in Angstroms. Lower values sample more grid points and generate finer visualizations but require more memory.

## 5.33 ContourTypedAddScoped

```
void ContourTypedAddScoped(unsigned int scope = BestScope)
```

Add a single contour to all grids of type *type* in *scope*.

## 5.34 ContourTypedCountScoped

```
int ContourTypedCountScoped(unsigned int scope = BestScope)
```

Return the number of contours of all grids of type *type* in *scope*. This essentially returns the number of contours of the first grid found.

## 5.35 ContourTypedMaxScoped

```
float ContourTypedMaxScoped(unsigned int scope = BestScope)
```

Returns the maximum allowable threshold for the first grid found of type *type* in *scope*.

## 5.36 ContourTypedMinScoped

```
float ContourTypedMinScoped(unsigned int scope = BestScope)
```

Returns the minimum allowable threshold for the first grid found of type *type* in *scope*.

## 5.37 ContourTypedRemoveScoped

```
void ContourTypedRemoveScoped(unsigned int scope = BestScope)
```

Remove the last contour from all grids of type *type* in *scope*.

## 5.38 ContourTypedSetLevelForIndexScoped

```
void ContourTypedSetLevelForIndexScoped(unsigned int index, float thresh,
                                        unsigned int scope = BestScope)
```

Set the contour *level* for the contour at *index* for all grids of type *type* in *scope*.

## 5.39 ContourVolume

```
float ContourVolume(unsigned int id, unsigned int contourIndex)
```

Return the enclosed volume of Grid *id*'s contour at index *contourIndex*.

## 5.40 Delete

```
void Delete(unsigned int id)
void Delete(const std::vector<unsigned int> &)
void Delete(const std::vector<OEPropDB::OEKey> &)
void Delete(const OEPropDB::OEKey &, bool all=false)
```

Deletes the specified object(s) from the current session. It is important to note that this does NOT delete the actual files (or entries within files) associated with the specified objects.

## 5.41 DeleteAll

```
void DeleteAll(bool force=false)
```

Clears the application back to a clean state with nothing loaded.

## 5.42 DeleteScoped

```
void DeleteScoped(unsigned int scope, bool inScope)
```

Scoped version of `Delete` but with the additional option to specify whether to delete everything within the scope or everything not within the scope using the *inScope* parameter.

## 5.43 FindByDataScoped

```
std::vector<OEPropDB::OEKey> FindByDataScoped(const std::string &query,
                                              unsigned int scope = AllScope)
```

Reserved for future implementation.

## 5.44 FindByQueryScoped

```
std::vector<OEPropDB::OEKey> FindByQueryScoped(const OEPropDB::OEKey &query,
                                               unsigned int scope = AllScope)
std::vector<OEPropDB::OEKey> FindByQueryScoped(const OEPropDB::OEKey &query,
                                               unsigned int atoms,
                                               unsigned int bonds,
                                               unsigned int scope = AllScope)
```

Uses the specified molecule as a query to search the specified scope. Returns a list of keys corresponding to molecules that match the specified query.

## 5.45 FindBySMARTSScoped

```
std::vector<OEPropDB::OEKey> FindBySMARTSScoped(const std::string &smarts,
                                                unsigned int scope = AllScope)
```

Uses the specified SMARTS pattern as a query to search the specified scope. Returns a list of keys corresponding to molecules that match the specified SMARTS pattern.

## 5.46 FindBySimilarityScoped

```
std::vector<OEPropDB::OEKey> FindBySimilarityScoped(const OEPropDB::OEKey &query,
                                                    float similarity,
                                                    unsigned int scope = AllScope)
```

Searches the specified scope for molecules with a Lingo similarity greater than the specified *similarity* cutoff with the specified molecule *query*.

## 5.47 FindByTitleScoped

```
std::vector<OEPropDB::OEKey> FindByTitleScoped(const std::string &expr,
                                               unsigned int scope = AllScope)
```

Uses the specified title query as a regular expression to search the specified scope for molecules whose title match the expression.

## 5.48 FindInRepository

```
void FindInRepository()
```

This function prompts the user to specify a query to search the contents of the application repository. The user then has the option to select some, none, or all of the hits to be placed in a new list.

## 5.49 FindOnDisk

```
void FindOnDisk()
```

This function prompts the user to specify a query to search a specific region of the user's hard drive for molecules matching the query. The user then has the option to load some, none, or all of the files containing the hits. This function will load the entire file where a hit was discovered as opposed to just the hits in that file.

## 5.50 GridAdd

```
unsigned int GridAdd(OESystem::OESkewGrid &grd, unsigned int listID = 0)
unsigned int GridAdd(OESystem::OEScalarGrid &grd, unsigned int listID = 0)
```

Adds the specified Python grid object to VIDA. If the optional listID argument is provided, then the grid is added to the specified list. Otherwise, a new list is created and the grid is added to it.

## 5.51 GridCheckIn

```
bool GridCheckIn(OESystem::OESkewGrid &grd,
                 unsigned int checkInType = OECheckInType_UnknownChange)
bool GridCheckIn(OESystem::OEScalarGrid &grd,
                 unsigned int checkInType = OECheckInType_UnknownChange)
```

Checks the specified Python accessible grid back into the main VIDA application which will synchronize any changes made to the grid in Python with the grid in VIDA. The grid in question must have been checked out of VIDA prior to being checked back in.

The *checkInType* parameter tells VIDA what type of changes were made to the grid in order to optimize the check in process. There are multiple allowable values for this parameter which can be OR'd together if necessary:

- OECheckInType_ConformerChange
- OECheckInType_CoordinateChange
- OECheckInType_DataChange
- OECheckInType_GraphChange
- OECheckInType_NoChange
- OECheckInType_RenderChange
- OECheckInType_AnnotationChange
- OECheckInType_UnknownChange

## 5.52 GridCheckOut

```
bool GridCheckOut(OESystem::OESkewGrid &grd, unsigned int id)
bool GridCheckOut(OESystem::OEScalarGrid &grd, unsigned int id)
```

Checks out a copy of the grid specified by *id* into the specified Python grid object (*grd*). Returns whether or not the check out process succeeded.

The checked out grid can be modified in Python, but those changes will not be applied to the original grid in VIDA until the modified grid is checked back in using the `GridCheckIn` command.

## 5.53 GridClear

```
void GridClear(unsigned int id)
```

Clears the data on the specified grid.

## 5.54 GridCopy

```
unsigned int GridCopy(unsigned int id)
```

Creates a copy of the specified grid and returns the ID of the newly created copy.

## 5.55 GridCreateElectrostaticsGrid

```
unsigned int GridCreateElectrostaticsGrid(OEPropDB::OEKey k,
                                          bool property=false)
```

Create an electrostatics grid for molecule *id*.

If *property* is true, then this grid is attached to *id* and does not show up in the list window. These grids cannot be saved to disk.

If *property* is false, then this grid is created as a separate entity from *id*.

## 5.56 GridCreateGaussian

```
unsigned int GridCreateGaussian(const OEPropDB::OEKey &k, float res)
```

Creates a gaussian grid representation for the molecule associated with the specified key at the specified resolution.

## 5.57 GridCreateGaussianProduct

```
unsigned int GridCreateGaussianProduct(const OEPropDB::OEKey &k, float res)
```

Creates a gaussian product grid representation for the molecule associated with the specified key at the specified resolution.

## 5.58 GridDefaultContourLevelByIndexGet

```
float GridDefaultContourLevelByIndexGet(unsigned int gridType,
                                        unsigned int index)
```

Returns the default contour level for contour *index* for grids of type *gridType*.

## 5.59 GridDefaultContourLevelByIndexSet

```
void GridDefaultContourLevelByIndexSet(unsigned int gridType,
                                       unsigned int index, float thresh)
```

Set the default contour level *thresh* for contour *index* for grids of type *gridType*.

## 5.60 GridDefaultNumContoursGet

```
unsigned int GridDefaultNumContoursGet(unsigned int gridType)
```

Returns the default number of contours for grids of type *gridType*.

## 5.61 GridDefaultNumContoursSet

```
void GridDefaultNumContoursSet(unsigned int gridType, unsigned int count)
```

Set the default number of contours for grids of type *gridType* to *count*.

## 5.62 GridInitializeContours

```
void GridInitializeContours(unsigned int id, bool update=true)
```

Initialize a grid to its default contours, colors and levels.

## 5.63 GridNormalize

```
bool GridNormalize(unsigned int id)
```

Normalizes a grid. This finds the sigma for the grid and divides all grid points by that value.

## 5.64 GridRegularize

```
    unsigned int GridRegularize(unsigned int id)
```

Regularizes the specified skew grid. Returns the ID of the newly created regular scalar grid.

## 5.65 GridToGaussianGrid

```
unsigned int GridToGaussianGrid(unsigned int id)
```

Creates a gaussian grid representation for the specified grid.

## 5.66 GridTypeGet

```
unsigned int GridTypeGet(unsigned int oerid)
```

Returns the type of grid for grid *id*.

## 5.67 GridTypeGetScoped

```
unsigned int GridTypeGetScoped(unsigned int scope = BestScope)
```

Returns the type of grid for the first grid found in *scope*.

## 5.68 GridTypeSet

```
void GridTypeSet(unsigned int id, unsigned int gridType)
```

Set the grid type of grid *id* to *gridType*.

## 5.69 GridTypeSetScoped

```
void GridTypeSetScoped(unsigned int gridType,
                       unsigned int scope = BestScope)
```

Scoped version of GridTypeSet.

## 5.70 GridWorkingGetScoped

```
unsigned int GridWorkingGetScoped(unsigned int scope = BestScope)
```

Returns the ID for the "working grid" which is determined in the following order: grid associated with selected contour, grid which is currently focused, grid which is a child object of the currently focused object, or lastly the first grid found in the current default scope.

## 5.71 HasGridChildrenScoped

```
int HasGridChildrenScoped(unsigned int scope = BestScope)
```

Returns whether or not any of the molecules in the specified scope have visible grids attached to them. Returns 0 for none of the molecules have visible grids, 1 for all molecules have visible grid, and 2 for at least one has a visible grid.

## 5.72 HasSurfaceChildrenScoped

```
int HasSurfaceChildrenScoped(const std::string &type,
                             unsigned int scope = BestScope)
```

Returns whether or not any of the molecules in the specified scope have visible surfaces attached to them. Returns 0 for none of the molecules have visible surfaces, 1 for all molecules have visible surface, and 2 for at least one has a visible surface.

## 5.73 Initialize

```
void Initialize(unsigned int id)
void Initialize(const OEPropDB::OEKey &)
```

Initializes the specified object for display. This function is automatically called on objects that are loaded into VIDA and ordinarily should not need to be called by the user.

## 5.74 IsAGrid

```
bool IsAGrid(unsigned int id)
bool IsAGrid(const OEPropDB::OEKey &k)
```

Returns whether or not the specified object is a grid.

## 5.75 IsAList

```
bool IsAList(unsigned int id)
```

Returns whether or not the specified object is a list.

## 5.76 IsAMolecule

```
bool IsAMolecule(unsigned int id)
bool IsAMolecule(const OEPropDB::OEKey &k)
```

Returns whether or not the specified object is a molecule.

## 5.77 IsAReflection

```
bool IsAReflection(unsigned int id)
bool IsAReflection(const OEPropDB::OEKey &k)
```

Returns whether or not the specified object is a reflection.

## 5.78 IsASmallMolecule

```
bool IsASmallMolecule(unsigned int id)
bool IsASmallMolecule(const OEPropDB::OEKey &k)
```

Returns whether or not the specified object is a small molecule.

## 5.79 IsASurface

```
bool IsASurface(unsigned int id)
bool IsASurface(const OEPropDB::OEKey &k)
```

Returns whether or not the specified object is a surface.

## 5.80 KeyGet

```
OEPropDB::OEKey KeyGet(const OESystem::OEBase &)
OEPropDB::OEKey KeyGet(const OESystem::OEBase &, const OEPropDB::OEKey &pkey)
```

Returns the key associated with the specified OEBase object. If the object's parent key is known, that can be passed as a parameter as well to aid the key determination process.

## 5.81 KeyIDGet

```
unsigned int KeyIDGet(const OEPropDB::OEKey &)
```

Returns the ID associated with the specified key.

## 5.82 KeyParentIDGet

```
unsigned int KeyParentIDGet(const OEPropDB::OEKey &)
```

Returns the ID of the parent object associated with the specified key. This function should be used for child object types such as atoms, bonds, triangles, and vertices.

## 5.83 KeySourceIDGet

```
unsigned int KeySourceIDGet(const OEPropDB::OEKey &)
```

Returns the ID of the parent object associated with a specified key. This function should be used for top-level object types such as molecules, grids, and surfaces in situations where these objects have been added as children to another object.

## 5.84 KeyTypeGet

```
std::string KeyTypeGet(const OEPropDB::OEKey &)
```

Returns a string representation of the object type associated with the specified key. Return values include:

- "Atom"
- "Bond"
- "Grid"
- "Mol"
- "Surface"
- "Triangle"
- "Vertex"

## 5.85 KeysGet

```
std::vector<OEPropDB::OEKey> KeysGet(unsigned int id)
```

Returns a list of all the keys for the specified ID. For most objects, the list returned will contain a single entry; however, for multi-conformer molecules, the list will include a key for each individual conformer.

## 5.86 ListAddObject

```
bool ListAddObject(unsigned int listid, unsigned int objectid)
```

Adds the object specified by *objectid* to the list specified by *listid*.

## 5.87 ListAddObjects

```
bool ListAddObjects(unsigned int listid,
                    const std::vector<unsigned int> &ids)
```

Adds the objects specified by the list of IDs (*ids*) to the list specified by *listid*.

## 5.88 ListGetNames

```
std::vector<std::string> ListGetNames()
```

Returns a list of the names of all the current lists.

## 5.89 ListGetObjectLists

```
std::vector<unsigned int> ListGetObjectLists(unsigned int objectid)
```

Returns a list of all the lists to which the object specified by *objectid* belongs.

## 5.90 ListGetObjects

```
std::vector<unsigned int> ListGetObjects(unsigned int listid)
```

Returns a list of IDs of all the objects contained within the list specified by *listid*.

## 5.91 ListMoveObject

```
bool ListMoveObject(unsigned int srclist, unsigned int dstlist,
                    unsigned int objectid)
```

Moves the object specified by *objectid* from the list specified by *srclist* to the list specified by *dstlist*.

## 5.92 ListMoveObjects

```
bool ListMoveObjects(unsigned int srclist, unsigned int dstlist,
                     const std::vector<unsigned int> &ids)
```

Moves the objects specified by list of IDs (*ids*) from the list specified by *srclist* to the list specified by *dstlist*.

## 5.93 ListNew

```
unsigned int ListNew(const std::string &name)
unsigned int ListNew(const std::string &name,
                     const std::vector<unsigned int> &ids)
unsigned int ListNew(const std::string &name,
                     const std::vector<OEPropDB::OEKey> &keys)
```

Creates a new list with the specified name. If a list of IDs or keys is specified with this call, all of the objects corresponding to those IDs or keys will be added to the list.

Returns the ID of the created list.

## 5.94 ListNewAnd

```
unsigned int ListNewAnd(const std::vector<unsigned int> &ids)
```

Creates a new list containing all of the objects that each appear in every list specified by the list of IDs (*ids*).

Returns the ID of the created list.

## 5.95 ListNewMarked

```
unsigned int ListNewMarked(const std::string &name="")
```

Creates a new list containing all the currently marked objects. Objects will not be removed from this list if they are subsequently unmarked.

Returns the ID of the created list.

## 5.96 ListNewOr

```
unsigned int ListNewOr(const std::vector<unsigned int> &ids)
```

Creates a new list containing all of the objects in all of the lists specified by the list of IDs (*ids*).

Returns the ID of the created list.

## 5.97 ListNewXor

```
unsigned int ListNewXor(const std::vector<unsigned int> &ids)
```

Creates a new list containing all of the objects that each appear only once among all the lists specified by the list of IDs (*ids*).

Returns the ID of the created list.

## 5.98 ListRemoveObject

```
bool ListRemoveObject(unsigned int listid, unsigned int objectid)
```

Removes the object specified by *objectid* from the list specified by *listid*.

## 5.99 ListRemoveObjects

```
bool ListRemoveObjects( unsigned int listid,
                        const std::vector<unsigned int> &ids )
```

Removes the object(s) specified by the list of IDs (*ids*) from the list specified by *listid*.

## 5.100 ListRootList

```
unsigned int ListRootList()
```

Returns the ID of the root list.

## 5.101 ListSubsetMarked

```
unsigned int ListSubsetMarked(unsigned int id, bool match)
```

Creates a new list containing a subset of the objects contained within the list specified by *id*. If the *match* parameter is True, the subset will contain all of the marked objects in the source list, if *match* is False, the subset will contain all of the non-marked objects in the source list.

Returns the ID of the created list.

## 5.102 ListSubsetQuery

```
unsigned int ListSubsetQuery(unsigned int id, const std::string &, bool match)
```

Creates a new list containing a subset of the objects contained within the list specified by *id*. If the *match* parameter is True, the subset will contain all of the objects that match the specified query in the source list, if *match* is Fale, the subset will contain all of the objects which fail to match the specified query in the source list. The query parameter can be specified as a SMARTS, a SMILES, or an IUPAC name.

Returns the ID of the created list.

## 5.103 MoleculeAdd

```
unsigned int MoleculeAdd(const std::string &, unsigned int listID = 0)
unsigned int MoleculeAdd(OEChem::OEMCMolBase &, unsigned int listID  = 0)
```

Adds a new molecule to VIDA. There are two implementations of this function. The first implementation expects a hexadecimal encoded OEB string specifying a molecule. The second implementation expects a Python molecule object.

If the optional listID argument is provided, then the molecule is added to the specified list. Otherwise, a new list is created and the molecule is added to it.

Returns the ID assigned to the newly added molecule or zero if the add failed for any reason.

## 5.104 MoleculeCheckIn

```
bool MoleculeCheckIn(OEChem::OEMCMolBase &mol,
                     unsigned int checkInType = OECheckInType_UnknownChange )
```

Checks the specified Python accessible molecule back into the main VIDA application which will synchronize any changes made to the molecule in Python with the molecule in VIDA. The molecule in question must have been checked out of VIDA prior to being checked back in.

The *checkInType* parameter tells VIDA what type of changes were made to the molecule in order to optimize the check in process. There are multiple allowable values for this parameter which can be OR'd together if necessary:

- OECheckInType_ConformerChange
- OECheckInType_CoordinateChange
- OECheckInType_DataChange
- OECheckInType_GraphChange

- OECheckInType_NoChange
- OECheckInType_RenderChange
- OECheckInType_AnnotationChange
- OECheckInType_UnknownChange

## 5.105 MoleculeCheckOut

```
bool MoleculeCheckOut(OEChem::OEMCMolBase &mol, unsigned int id)
```

Checks out a copy of the molecule specified by *id* into the specified Python molecule object (*mol*). Returns whether or not the check out process succeeded.

The checked out molecule can be modified in Python, but those changes will not be applied to the original molecule in VIDA until the modified molecule is checked back in using the `MoleculeCheckIn` command.

## 5.106 MoleculeComponentNamesGet

```
std::vector<std::string> MoleculeComponentNamesGet(unsigned int id)
```

Returns a list of names of the individual components of the molecule specified by `id` based on PDB related subsets.

## 5.107 MoleculeExamine

```
bool MoleculeExamine(OEChem::OEMCMolBase &, unsigned int id)
bool MoleculeExamine(OEChem::OEMCMolBase &, const OEPropDB::OEKey &key)
```

Copies the molecule associated with the specified ID or key into the specified Python molecule object. Returns True if the copy happened successfully. In cases where a script needs read-only access to a molecule, MoleculeExamine provides a much faster way to retrieve molecules than MoleculeCheckOut or MoleculeGet. Molecules retrieved using MoleculeExamine cannot be checked back in using MoleculeCheckIn.

## 5.108 MoleculeGet

```
bool MoleculeGet(OEChem::OEMCMolBase &, unsigned int id)
bool MoleculeGet(OEChem::OEMCMolBase &, const OEPropDB::OEKey &key)
OEChem::OEMol MoleculeGet(unsigned int id)
OEChem::OEMol MoleculeGet(const OEPropDB::OEKey &key)
```

Copies the molecule associated with the specified ID or key into the specified Python molecule object. Returns whether or not the copy happened successfully.

## 5.109 MoleculeHasComponents

```
bool MoleculeHasComponents(unsigned int id)
```

Returns whether or not the molecule specified by `id` contains multiple distinct components.

## 5.110 MoleculeMaxResidueGet

```
int MoleculeMaxResidueGet(unsigned int id)
```

Returns the largest residue index of all the residues contained within the specified molecule.

## 5.111 MoleculeMergeScoped

```
unsigned int MoleculeMergeScoped(unsigned int scope = BestScope,
                                 const std::string &name="Merged",
                                 unsigned int listid=0)
```

Merges all of the molecules in the specified scope into a single new molecule. The `name` parameter specifies the new molecule's name and the `listid` parameter specifies in which list to place this new molecule. If `listid` is 0, a new list will be created to contain the created molecule.

## 5.112 MoleculeNewSubset

```
std::vector<unsigned int> MoleculeNewSubset(unsigned int listid,
                                            unsigned int id, bool pdb)
std::vector<unsigned int> MoleculeNewSubset(unsigned int listid,
                                            unsigned int id, unsigned int scope,
                                            bool unmatched)
std::vector<unsigned int> MoleculeNewSubset(unsigned int listid,
                                            unsigned int id,
                                            const std::vector<std::vector<unsigned int> > &parts)
```

This function creates a series of new molecules from the given molecule based on its internal components determined by OEChem using the OEDetermineComponents function.

## 5.113 MoleculeNewSubsetScoped

```
std::vector<unsigned int> MoleculeNewSubsetScoped(unsigned int listid,
                                                  unsigned int scope, bool pdb)
std::vector<unsigned int> MoleculeNewSubsetScoped(unsigned int listid,
                                                  unsigned int mscope,
                                                  unsigned int scope,
                                                  bool unmatched)
```

This function creates a series of new molecules from the given scope based on its internal components determined by OEChem using the OEDetermineComponents function.

## 5.114 MoleculeResidueNameSetScoped

```
void MoleculeResidueNameSetScoped(const std::string &name,
                                  unsigned int scope = BestScope)
```

Renames all the residues in the specified scope.

## 5.115 MoleculeResidueSet

```
int MoleculeResidueSet(unsigned int id, int residueNum,
                       bool incrementPerRes=false)
```

Sets the residue number of every residue in the specified molecule to the specified value (*residueNum*). The *incrementPerRes* parameter is currently reserved for future expansion.

## 5.116 MoleculeSetProperty

```
void MoleculeSetProperty(const std::vector<OEPropDB::OEKey> &keys,
                         unsigned char action, std::string prop, bool state)
```

Reserved for internal use.

## 5.117 MoleculeSizeCutoffGet

```
unsigned int MoleculeSizeCutoffGet()
```

Returns the atom number cutoff at which point a molecule is considered to be a "large" molecule. The default is 255.

## 5.118 MoleculeSizeCutoffSet

```
void MoleculeSizeCutoffSet(unsigned int size)
```

Sets the atom number cutoff at which point a molecule is considered to be a "large" molecule. The default is 255.

## 5.119 MoleculeUpdate

```
bool MoleculeUpdate(OEChem::OEMCMolBase &)
```

Updates the original molecule stored in VIDA with any changes that may have been made to the specified Python molecule.

## 5.120 NameGet

```
std::string NameGet(unsigned int)
std::string NameGet(const OEPropDB::OEKey &)
```

Returns the name of the object specified by either its ID or key.

## 5.121 NameSet

```
void NameSet(unsigned int, const std::string &)
void NameSet(const OEPropDB::OEKey &, const std::string &)
```

Sets the name of the object specified by either its ID or key.

## 5.122 OEFuseKeyGroups

```
OEKeyGroup OEFuseKeyGroups(const std::vector<OEKeyGroup> &k1,
                           unsigned char type)
```

Fuses the specified keys groups into a single key group.

## 5.123 OEKeyIterToVector

```
std::vector<OEPropDB::OEKey>
  OEKeyIterToVector(OESystem::OEIterBase<const OEPropDB::OEKey> *iterbase)
```

Converts a key iterator into a list of keys.

## 5.124 PropertyTypeGet

```
unsigned int PropertyTypeGet(const std::string &type)
```

Returns the integer property type for the string *type*. See KeyTypeGet for a list of types.

## 5.125 SDDataGet

```
std::string SDDataGet(unsigned int id, const std::string &tag)
std::string SDDataGet(const OEPropDB::OEKey &key, const std::string &tag)
```

Returns the SD data associated with the specified tag on the specified object.

## 5.126 SDDataHas

```
bool SDDataHas(unsigned int id, const std::string &tag)
bool SDDataHas(const OEPropDB::OEKey &key, const std::string &tag)
```

Returns whether or not the specified object contains SD data with the specified tag.

## 5.127 SDDataSet

```
void SDDataSet(unsigned int id, const std::string &tag, const std::string &data)
void SDDataSet(const OEPropDB::OEKey &key, const std::string &tag,
               const std::string &data)
```

Sets the value of the SD data associated with the specified tag on the specified object.

## 5.128 SurfaceAdd

```
unsigned int SurfaceAdd(OESpicoli::OESurface &surf, unsigned int listID = 0)
```

Adds a new Python surface object to VIDA. Returns the ID assigned to the newly added surface or zero if the add failed for any reason. If the optional listID argument is provided, then the surface is added to the specified list. Otherwise, a new list is created and the surface is added to it.

## 5.129 SurfaceBestFloodScoped

```
int SurfaceBestFloodScoped(unsigned int scope = BestScope)
```

Return the largest flood value for all surfaces in the given scope. A flood value defines the surface being scribed.

## 5.130 SurfaceCheckIn

```
bool SurfaceCheckIn(OESpicoli::OESurface &surf,
                    unsigned int checkInType = OECheckInType_UnknownChange)
```

Checks the specified Python accessible surface back into the main VIDA application which will synchronize any changes made to the surface in Python with the surface in VIDA. The surface in question must have been checked out of VIDA prior to being checked back in.

The *checkInType* parameter tells VIDA what type of changes were made to the surface in order to optimize the check in process. There are multiple allowable values for this parameter which can be OR'd together if necessary:

- OECheckInType_ConformerChange
- OECheckInType_CoordinateChange
- OECheckInType_DataChange
- OECheckInType_GraphChange
- OECheckInType_NoChange
- OECheckInType_RenderChange
- OECheckInType_AnnotationChange
- OECheckInType_UnknownChange

## 5.131 SurfaceCheckOut

```
bool SurfaceCheckOut(OESpicoli::OESurface &surf, unsigned int id)
```

Checks out a copy of the surface specified by *id* into the specified Python surface object (*surf*). Returns whether or not the check out process succeeded.

The checked out surface can be modified in Python, but those changes will not be applied to the original surface in VIDA until the modified surface is checked back in using the `SurfaceCheckIn` command.

## 5.132 SurfaceCreate

```
unsigned int SurfaceCreate(const std::string &, const OEPropDB::OEKey &key,
                           bool property=false)
```

Create a surface of type *type* based on the molecule represented by *key*.

If *property* is true, then make the surface a property surface that is bound to the molecule.

## 5.133 SurfaceCreateScoped

```
bool SurfaceCreateScoped(const std::string &, bool single_surface=false,
                         unsigned int scope = BestScope)
```

Create a surface of type *type* for each molecule in *scope*.

If *property* is true, then make the surface a property surface that is bound to the molecule.

## 5.134 SurfaceCropDistance

```
void SurfaceCropDistance(unsigned int id, float distance)
```

Retains the section of the surface specified by *oerid* to a distance *dist* from the selected set of atoms.

## 5.135 SurfaceCropDistanceFrom

```
void SurfaceCropDistanceFrom(const OEPropDB::OEKey &s, const OEPropDB::OEKey &,
                             float distance)
```

Retains the section of the surface specified by *s* based on its distance to another object. The reference object can be a molecule or another surface. The distance cutoff is measured in Angstroms.

## 5.136 SurfaceCropScribedScoped

```
void SurfaceCropScribedScoped(unsigned int scope = BestScope)
```

Retains the section of all surfaces in *scope* to a distance *dist* from the selected set of atoms.

## 5.137 SurfaceCropUnscribedScoped

```
void SurfaceCropUnscribedScoped(unsigned int scope = BestScope)
```

Remove the scribed sections of all scribed surfaces in *scope*.

## 5.138 SurfaceDelete

```
void SurfaceDelete(unsigned int id)
```

Delete the surface specified by *id*.

## 5.139 SurfaceGenerateBox

```
unsigned int SurfaceGenerateBox(float x, float y, float z, float width,
                                float height, float depth)
```

Creates and returns the id of a surface representing a box as determined by the parameters *x*, *y*, *z*, *width*, *height*, and *depth*.

## 5.140 SurfaceGenerateSphere

```
unsigned int SurfaceGenerateSphere(float x, float y, float z, float radius,
                                   int level=10)
```

Creates and returns the id of a surface representing a sphere as determined by the parameters *x*, *y*, *z*, and *radius*. The *level* parameter specifies how many iterations should be used in determining the quality of the surface.

## 5.141 SurfaceGenerateSpline

```
unsigned int SurfaceGenerateSpline(const std::vector<float> &coords,
                                   int splineResolution, int seq_res,
                                   int cross_res, float width, float thick,
                                   int splineType, int style)
```

Creates and returns the id of a surface representing a spline as determined by the parameters *coords*, *splineResolution*, *seq_res*, *cross_res*, *width*, *thick*, *splineType*, and *style*.

## 5.142 SurfacePickTriangle

```
void SurfacePickTriangle(unsigned int oerid, const OEPropDB::OEKey &trikey,
                         bool addtriangle=false)
```

Pick a surface triangle specified by *trikey*. Ordinarily, this function does not need to be called.

## 5.143 SurfaceProbeRadiusGet

```
float SurfaceProbeRadiusGet()
```

Get the probe radius used when creating surfaces.

## 5.144 SurfaceProbeRadiusSet

```
float SurfaceProbeRadiusSet(float radius)
```

Set the probe radius to *radius* that is used when creating surfaces.

## 5.145 SurfaceResolutionGet

```
float SurfaceResolutionGet()
```

Get the resolution used when creating surfaces.

## 5.146 SurfaceResolutionSet

```
float SurfaceResolutionSet(float res)
```

Set the resolution used when creating surfaces. The resolution is set to *resolution*.

## 5.147 SurfaceRestoreScoped

```
void SurfaceRestoreScoped(unsigned int scope = BestScope)
```

Restore all surfaces in *scope* to their original states.

## 5.148 SurfaceScribeScoped

```
void SurfaceScribeScoped(int flood,
                         unsigned int scope = BestScope)
```

Scribe all scribed surfaces in scope to include all portions of the surface that fall within *flood*. Ordinarily, this function does not need to be called.

## 5.149 SurfaceSetPotentialFromGrid

```
void SurfaceSetPotentialFromGrid(unsigned int id, unsigned int gridid)
```

Sets the surface potential for the specified surface using the values in the specified grid.

## 5.150 SurfaceSetPotentialFromGridScoped

```
void SurfaceSetPotentialFromGridScoped(unsigned int gridid,
                                       unsigned int scope = BestScope)
```

Sets the surface potential for all the surfaces in the specified scope using the values in the specified grid.

## 5.151 SurfaceVolume

```
float SurfaceVolume(unsigned int id)
```

Returns the enclosed volume of surface *id*.

## 5.152 SymmetryNumOperators

```
unsigned int SymmetryNumOperators(unsigned int id)
```

This function returns the number of symmetry operators for the specified id.

## 5.153 SymmetryOperatorEnabledGet

```
bool SymmetryOperatorEnabledGet(unsigned int id, unsigned int operatorNumber)
```

## 5.154 SymmetryOperatorEnabledSet

```
bool SymmetryOperatorEnabledSet(unsigned int id, unsigned int operatorNumber,
                                bool enable)
```

## 5.155 SymmetryRadiusGet

```
float SymmetryRadiusGet()
```

## 5.156 SymmetryRadiusSet

```
float SymmetryRadiusSet(float r)
```

## 5.157 SymmetryRealize

```
unsigned int SymmetryRealize(unsigned int id, bool make_new_mol,
                             std::vector<float> center=std::vector<float>(),
                             float radius=0.0f)
```

## 5.158 XRayAutoMapCalculationPrefsSet

```
void XRayAutoMapCalculationPrefsSet()
void XRayAutoMapCalculationPrefsSet(const std::string &map1,
                                                const std::string &map2="")
```

Set the map types for maps which should be calculated automatically after reading a reflection file.

## 5.159 XRayCalculateMap

```
unsigned int XRayCalculateMap(unsigned int map_id,
                              unsigned int refln_id,
                              std::string map_type)
unsigned int XRayCalculateMap(unsigned int map_id,
                              unsigned int refln_id,
                              unsigned int map_type)
```

This set of overloaded functions calculates a map for the set of reflection data specified by 'refln_id'. If 'map_id' is non-zero, then the map replaces the grid with that ID. Otherwise, an entirely new map is created. The string 'map_type' parameter may be one of: Fo, Fc, 2Fo-Fc, Fo-Fc, 3Fo-2Fc, 5Fo-3Fc, Fo*Fo, or Fo*FOM, while the integer version may be one of: 0 Fo, 1 Fc, 2 TwoFoFc, 3 FoFc, 4 ThreeFoTwoFc, 5 FiveFoThreeFc, 8 FoSquared, 9 FoFom.

## 5.160 XRayCalculatePhases

```
bool XRayCalculatePhases(unsigned int refln_id,
                         unsigned int model_id)
```

This is experimental, and should not be called by the user.

## 5.161 XRayGetCell

```
std::vector<float> XRayGetCell(unsigned int id)
```

Returns the cell constants for the given object in a list, in the order a, b, c, Alpha, Beta, Gamma.

## 5.162 XRayGetSpaceGroup

```
std::string XRayGetSpaceGroup(unsigned int id)
```

Returns the spacegroup name for the given object.

## 5.163 XRayMTZColumnNamesCurrentDefaultsGet

```
std::vector<std::string> XRayMTZColumnNamesCurrentDefaultsGet()
```

## 5.164 XRayMTZColumnNamesGet

```
std::vector<std::string> XRayMTZColumnNamesGet()
bool XRayMTZColumnNamesGet(std::vector<std::string> &names)
```

Fills up the input list/vector with the MTZ file column names as specified by *XRayMTZColumnNamesSet*. Unless *XRayMTZColumnNamesSet* was called with the *permanent* flag, this function will only work once per call to *XRayMTZColumnNamesSet*.

This function returns True if it successfully filled out the list/vector, or False otherwise.

## 5.165 XRayMTZColumnNamesSet

```
void XRayMTZColumnNamesSet(const std::vector<std::string> &names,
                           bool permanent=false)
```

Stores a list of column names for mtz files. The next time an MTZ file is read, instead of prompting the user, the column names are taken from the list/vector passed to this function. The column names are in the order Fo, Phi, Fc, PhiC, Sigma, FOM, RFree. If the *permanent* flag is set to True, then these values are used instead of prompting until the program exits. If *permanent* is false, then the values are only used once.

In a script, there's no reason to call this command instead of pre-specifying the prompt results with *PromptResponseVectMulti*, as shows up in the journal file. However, if creating a function which will read an MTZ file directly then this command is needed since the *PromptResponse* commands don't work in interactive commands.

## 5.166 XRayMTZColumnNamesStandardDefaultsGet

```
std::vector<std::string> XRayMTZColumnNamesStandardDefaultsGet()
```

## 5.167 XRaySetCrystalParams

```
bool XRaySetCrystalParams(unsigned int id,
                          unsigned int from_object_id)
bool XRaySetCrystalParams(unsigned int id,
                          std::vector<float> cell,
                          unsigned int sg)
bool XRaySetCrystalParams(unsigned int id,
                          std::vector<float> cell,
                          const std::string &sgname)
bool XRaySetCrystalParams(unsigned int id, float a, float b,
                          float c, float alpha, float beta,
                          float gamma, unsigned int sg)
bool XRaySetCrystalParams(unsigned int id, float a, float b,
                          float c, float alpha, float beta,
                          float gamma, const std::string &sgname)
```

Set the crystal parameters for the given reflection. The parameters may be specified in multiple ways, which are obvious from the names of the arguments.

## 5.168 XRayValidMaptypes

```
std::string XRayValidMaptypes(unsigned int refln_id)
```

Not all map types are available for every reflection data set, since not all files have all required data for every possible map calculation. For example, a reflection data set without Fc data will not be able to calculate any map types which require Fc. This function returns (as a string), the valid map types for the given reflection data set. The string is a concatenation of all available map types, with each type separated by the 'l' character.

CHAPTER

SIX

# MOLECULE BUILDER FUNCTIONS

The functions detailed in this section provide the ability to create new molecules from basic inputs and to modify existing ones.

## 6.1 AtomAddHydrogens

```
void AtomAddHydrogens(const OEPropDB::OEKey &k, bool polaronly=false)
```

Adds explicit hydrogens to the specified atom. If *polaronly* is set to True, this will only add polar hydrogens to the specified atom.

## 6.2 AtomAddHydrogensScoped

```
void AtomAddHydrogensScoped(bool polaronly=false,
                            unsigned int scope = SelectedScope)
```

Scoped version of `AtomAddHydrogens`.

## 6.3 AtomAtomicNumDefaultGet

```
unsigned int AtomAtomicNumDefaultGet()
```

Returns the default value of the atomic number to be used when setting the atomic number of an atom via a mouse action.

## 6.4 AtomAtomicNumDefaultSet

```
void AtomAtomicNumDefaultSet(unsigned int)
```

Sets the default value of the atomic number to be used when setting the atomic number of an atom via a mouse action.

## 6.5 AtomAtomicNumGet

```
unsigned int AtomAtomicNumGet(const OEPropDB::OEKey &k)
```

Returns the atomic number of the specified atom.

## 6.6 AtomAtomicNumSet

```
void AtomAtomicNumSet(const OEPropDB::OEKey &k, unsigned int an)
```

Sets the atomic number of the specific atom.

## 6.7 AtomAtomicNumSetScoped

```
void AtomAtomicNumSetScoped(unsigned int an,
                            unsigned int scope = SelectedScope)
```

Scoped version of `AtomAtomicNumSet`.

## 6.8 AtomAttach

```
void AtomAttach(const OEPropDB::OEKey &k, const std::string &oeb)
void AtomAttach(const OEPropDB::OEKey &k1, const OEPropDB::OEKey &k2)
void AtomAttach(const OEPropDB::OEKey &k, const std::string &smiles, int attach)
```

Attaches the specified molecule fragment to the specified atom. There are three different implementations of this function which allow for different specifications of the fragment. The atom being attached to, is always the first parameter. The attachment is performed by the creation of a single bond.

The first implementation takes a hexadecimal encoding of a molecule specified in OEB format. This implementation is primarily intended for internal use.

The second implementation takes an OEKey parameter specifying an atom on an already loaded fragment to which the attachment will occur.

The third implementation takes a SMILES string representing the fragment to be attached as well as an index specifying which atom in that fragment will be the attachment point.

## 6.9 AtomDelete

```
void AtomDelete(const OEPropDB::OEKey &k)
```

Deletes the specified atom as well as any attached hydrogens.

## 6.10 AtomDeleteHydrogens

```
void AtomDeleteHydrogens(const OEPropDB::OEKey &k)
```

Deletes all of the hydrogens attached to the specified atom.

## 6.11 AtomDeleteHydrogensScoped

```
void AtomDeleteHydrogensScoped(bool nonpolar=false,
                               unsigned int scope = SelectedScope)
```

Scoped version of `AtomDeleteHydrogens`.

## 6.12 AtomDeleteScoped

```
void AtomDeleteScoped(unsigned int scope = SelectedScope)
```

Scoped version of `AtomDelete`.

## 6.13 AtomFormalChargeDefaultGet

```
int AtomFormalChargeDefaultGet()
```

Returns the default value of the formal to be used when setting the formal of an atom via a mouse action.

## 6.14 AtomFormalChargeDefaultSet

```
void AtomFormalChargeDefaultSet(int)
```

Sets the default value of the formal to be used when setting the formal of an atom via a mouse action.

## 6.15 AtomFormalChargeGet

```
int AtomFormalChargeGet(const OEPropDB::OEKey &k)
```

Returns the formal charge for the specified atom.

## 6.16 AtomFormalChargeModify

```
void AtomFormalChargeModify(const OEPropDB::OEKey &k, int delta)
```

Modifies the formal charge on the specified atom by the specified delta value. The delta value can be either positive or negative.

## 6.17 AtomFormalChargeModifyScoped

```
void AtomFormalChargeModifyScoped(int delta,
                                  unsigned int scope = SelectedScope)
```

Scoped version of `AtomFormalChargeModify`.

## 6.18 AtomFormalChargeSet

```
void AtomFormalChargeSet(const OEPropDB::OEKey &k, int chg)
```

Sets the formal charge on the specified atom.

## 6.19 AtomFormalChargeSetScoped

```
void AtomFormalChargeSetScoped(int chg,
                               unsigned int scope = SelectedScope)
```

Scoped version of `AtomFormalChargeSet`.

## 6.20 AtomFuse

```
void AtomFuse(const OEPropDB::OEKey &k, const std::string &oeb)
void AtomFuse(const OEPropDB::OEKey &k1, const OEPropDB::OEKey &k2)
void AtomFuse(const OEPropDB::OEKey &k, const std::string &smiles, int attach)
```

Fuses the specified molecule fragment to the specified atom. There are three different implementations of this function which allow for different specifications of the fragment. The atom being fused to is always the first parameter.

The first implementation takes a hexadecimal encoding of a molecule specified in OEB format. This implementation is primarily intended for internal use.

The second implementation takes an OEKey parameter specifying an atom on an already loaded fragment where the fusion will occur.

The third implementation takes a SMILES string representing the fragment to be fused as well as an index specifying which atom in that fragment will be the site of the fusion.

## 6.21 AtomHybridizationGet

```
unsigned int AtomHybridizationGet(const OEPropDB::OEKey &k)
```

Returns the hybridization state of the specified atom.

## 6.22 AtomHybridizationSet

```
void AtomHybridizationSet(const OEPropDB::OEKey &k, unsigned int hyb)
```

Sets the hybridization state of the specified atom.

## 6.23 AtomHybridizationSetScoped

```
void AtomHybridizationSetScoped(unsigned int hyb,
                                unsigned int scope = SelectedScope)
```

Scoped version of `AtomHybridizationSet`.

## 6.24 AtomIsotopeGet

```
unsigned int AtomIsotopeGet(const OEPropDB::OEKey &k)
```

Returns the isotope of the specified atom.

## 6.25 AtomIsotopeSet

```
void AtomIsotopeSet(const OEPropDB::OEKey &k, unsigned int iso)
```

Sets the isotope for the specified atom.

## 6.26 AtomIsotopeSetScoped

```
void AtomIsotopeSetScoped(unsigned int iso,
                          unsigned int scope = SelectedScope)
```

Scoped version of `AtomIsotopeSet`.

## 6.27 AtomSprout

```
void AtomSprout(const OEPropDB::OEKey &k, unsigned int elem=6,
                unsigned int order=1)
```

Sprouts a new atom off of the specified atom. The atomic number of the new atom is specified by *elem* and the order of the bond to the new atom is specified by the *order* parameter.

## 6.28 AtomSproutScoped

```
void AtomSproutScoped(unsigned int elem=6, unsigned int order=1,
                      unsigned int scope = SelectedScope)
```

Scoped version of `AtomSprout`.

## 6.29 AtomStereoDefaultGet

```
std::string AtomStereoDefaultGet()
```

Returns the default value for the stereochemistry setting to be used when setting the stereochemistry via a mouse action. Allowed values include:

- "R"
- "S"
- "Invert"

## 6.30 AtomStereoDefaultSet

```
void AtomStereoDefaultSet(const std::string &s)
```

Sets the default value for the stereochemistry setting to be used when setting the stereochemistry via a mouse action. Allowed values include:

- "R"
- "S"
- "Invert"

## 6.31 AtomStereoSet

```
void AtomStereoSet(const OEPropDB::OEKey &k, const std::string &s)
```

Sets the CIP stereochemistry of the specified atom. Allowed values are "R" and "S".

## 6.32 AtomStereoSetScoped

```
void AtomStereoSetScoped(const std::string &s,
                         unsigned int scope = SelectedScope)
```

Scoped version of `AtomStereoSet`.

## 6.33 AtomStereoToggle

```
void AtomStereoToggle(const OEPropDB::OEKey &k)
```

Toggles the stereochemistry of the specified atom.

## 6.34 AtomStereoToggleScoped

```
void AtomStereoToggleScoped(unsigned int scope = SelectedScope)
```

Scoped version of: `AtomStereoToggle`.

## 6.35 BondAngleGet

```
double BondAngleGet(const OEPropDB::OEKey &k1, const OEPropDB::OEKey &k2)
double BondAngleGet(const OEPropDB::OEKey &k1, const OEPropDB::OEKey &k2,
                    const OEPropDB::OEKey &k3)
```

Returns the angle between the two specified bonds or the three specified atoms.

## 6.36 BondAngleModify

```
void BondAngleModify(const OEPropDB::OEKey &k1, const OEPropDB::OEKey &k2,
                     double angle, unsigned int rmode)
void BondAngleModify(const OEPropDB::OEKey &k1, const OEPropDB::OEKey &k2,
                     const OEPropDB::OEKey &k3, double ang, unsigned int rmode)
```

Modifies the angle between the two specified bonds or the three specified atoms. The amount by which the angle is adjusted is specified by the *angle* parameter. The portion of the molecule which is actually moved in this process is specified by the *rmode* parameter:

- 0 - the smaller portion of the molecule will be moved
- 1 - the larger portion of the molecule will be moved

## 6.37 BondAngleSet

```
void BondAngleSet(const OEPropDB::OEKey &k1, const OEPropDB::OEKey &k2,
                  double angle, unsigned int rmode)
void BondAngleSet(const OEPropDB::OEKey &k1, const OEPropDB::OEKey &k2,
                  const OEPropDB::OEKey &k3, double ang, unsigned int rmode)
```

Sets the angle between the two specified bonds or the three specified atoms. The desired angle is specified by the *angle* parameter. The portion of the molecule which is actually moved in this process is specified by the *rmode* parameter:

- 0 - the smaller portion of the molecule will be moved
- 1 - the larger portion of the molecule will be moved

## 6.38 BondCreate

```
void BondCreate(const OEPropDB::OEKey &k1, const OEPropDB::OEKey &k2,
                unsigned int order=1, double length=0.0)
```

Creates a new bond between the two specified atoms with an order and length specified by the two eponymous parameters.

## 6.39 BondDelete

```
void BondDelete(const OEPropDB::OEKey &k)
```

Deletes the specified bond.

## 6.40 BondDeleteScoped

```
void BondDeleteScoped(unsigned int scope = SelectedScope)
```

Scoped version of **:oefunc:'BondDelete#VFCommands::BondDelete'**.

## 6.41 BondFuse

```
void BondFuse(const OEPropDB::OEKey &b, const std::string &oeb)
void BondFuse(const OEPropDB::OEKey &b1, const OEPropDB::OEKey &b2)
void BondFuse(const OEPropDB::OEKey &b, const OEPropDB::OEKey &e,
              const std::string &oeb)
void BondFuse(const OEPropDB::OEKey &b, const std::string &smiles, int battach,
              int eattach)
void BondFuse(const OEPropDB::OEKey &b1, const OEPropDB::OEKey &e1,
              const OEPropDB::OEKey &b2, const OEPropDB::OEKey &e2)
void BondFuse(const OEPropDB::OEKey &beg, const OEPropDB::OEKey &end,
              const std::string &smiles, int battach, int eattach)
```

Fuses two bonds together. There are multiple implementations of this function which allow for different mechanisms of specifying the source bond as well as the target bond (and associated fragment). The source bond can be specified using a distinct bond key or by specifying the keys of the beginning and end atoms of that bond. The target bond can also be specified using a distinct bond key or by two keys representing the beginning and end atoms of the bond. However, the target bond can also be set as part of a specified fragment.

One implementation allows for specification of a hexadecimal encoded OEB string representing the molecule fragment and selected target bond. This method is primarily intended for internal use.

The other mechanism allows for specification of the fragment as a SMILES string associated with an index for the beginning and end atoms defining the bond.

## 6.42 BondLengthGet

```
double BondLengthGet(const OEPropDB::OEKey &k)
double BondLengthGet(const OEPropDB::OEKey &k1, const OEPropDB::OEKey &k2)
```

Returns the length of the specified bond. The bond can be specified using either a bond key or two atom keys corresponding to the endpoints of the bond.

## 6.43 BondLengthModify

```
void BondLengthModify(const OEPropDB::OEKey &k, double len, unsigned int rmode)
void BondLengthModify(const OEPropDB::OEKey &k1, const OEPropDB::OEKey &k2,
                      double length, unsigned int rmode)
```

Modifies the length of the specified bond by the amount specified in the *len* parameter. The bond can be specified using either a bond key or two atom keys corresponding to the endpoints of the bond. The portion of the molecule which is actually moved is determined by the *rmode* parameter. Available values include:

- 0 - the smaller portion of the molecule is moved
- 1 - the larger portion of the molecule is moved

## 6.44 BondLengthSet

```
void BondLengthSet(const OEPropDB::OEKey &k, double len, unsigned int rmode)
void BondLengthSet(const OEPropDB::OEKey &k1, const OEPropDB::OEKey &k2,
                   double length, unsigned int rmode)
```

Sets the length of the specified bond to the value specified in the *len* parameter. The bond can be specified using either a bond key or two atom keys corresponding to the endpoints of the bond. The portion of the molecule which is actually moved is determined by the *rmode* parameter. Available values include:

- 0 - the smaller portion of the molecule is moved
- 1 - the larger portion of the molecule is moved

## 6.45 BondLengthSetScoped

```
void BondLengthSetScoped(double length,
                         unsigned int rmode=OEForge::RotateMode::Smaller,
                         unsigned int scope = SelectedScope)
```

Scoped version of `BondLengthSet`.

## 6.46 BondOrderDefaultGet

```
unsigned int BondOrderDefaultGet()
```

Returns the default value of the bond order used when setting the order of a bond through a mouse action.

## 6.47 BondOrderDefaultSet

```
void BondOrderDefaultSet(unsigned int)
```

Sets the default value of the bond order used when setting the order of a bond through a mouse action.

## 6.48 BondOrderGet

```
unsigned int BondOrderGet(const OEPropDB::OEKey &k)
unsigned int BondOrderGet(const OEPropDB::OEKey &k1, const OEPropDB::OEKey &k2)
```

Returns the order of the specified bond. The bond can be specified as a distinct bond key or two keys corresponding to the endpoint atoms.

## 6.49 BondOrderSet

```
void BondOrderSet(const OEPropDB::OEKey &k, unsigned int order)
void BondOrderSet(const OEPropDB::OEKey &k1, const OEPropDB::OEKey &k2,
                  unsigned int order)
```

Sets the order of the specified bond. The bond can be specified as a distinct bond key or two keys corresponding to the endpoint atoms.

## 6.50 BondOrderSetScoped

```
void BondOrderSetScoped(unsigned int order, unsigned int scope)
```

Scoped version of `BondOrderSet`.

## 6.51 BondStereoDefaultGet

```
std::string BondStereoDefaultGet()
```

Returns the default value for bond stereochemistry setting to be used when setting the stereochemistry via a mouse action. Allowed values include:

- "E"
- "Z"
- "Invert"

## 6.52 BondStereoDefaultSet

```
void BondStereoDefaultSet(const std::string &)
```

Sets the default value for bond stereochemistry setting to be used when setting the stereochemistry via a mouse action. Allowed values include:

- "E"
- "Z"
- "Invert"

## 6.53 BondStereoSet

```
void BondStereoSet(const OEPropDB::OEKey &k, const std::string &s)
void BondStereoSet(const OEPropDB::OEKey &k1, const OEPropDB::OEKey &k2,
                   const std::string &s)
```

Sets the CIP stereochemistry for the specified bond. The bond can be specified as a distinct bond key or by two atom keys corresponding to the endpoint atoms. Allowed values for stereochemistry include:

- "E"
- "Z"

## 6.54 BondStereoSetScoped

```
void BondStereoSetScoped(const std::string &s,
                         unsigned int scope = SelectedScope)
```

Scoped version of `BondStereoSet`.

## 6.55 BondStereoToggle

```
void BondStereoToggle(const OEPropDB::OEKey &k)
void BondStereoToggle(const OEPropDB::OEKey &k1, const OEPropDB::OEKey &k2)
```

Toggles the stereochemistry for the specified bond. The bond can be specified as a distinct bond key or by two atom keys corresponding to the endpoint atoms.

## 6.56 BondStereoToggleScoped

```
void BondStereoToggleScoped(unsigned int scope = SelectedScope)
```

Scoped version of `BondStereoToggle`.

## 6.57 BondTorsionGet

```
double BondTorsionGet(const OEPropDB::OEKey &k1, const OEPropDB::OEKey &k2,
                      const OEPropDB::OEKey &k3)
double BondTorsionGet(const OEPropDB::OEKey &k1, const OEPropDB::OEKey &k2,
                      const OEPropDB::OEKey &k3, const OEPropDB::OEKey &k4)
```

Returns the torsion (dihedral) angle specified by either three bonds or four atoms. The order of specification is important in the determination of the angle.

## 6.58 BondTorsionModify

```
void BondTorsionModify(const OEPropDB::OEKey &k1, const OEPropDB::OEKey &k2,
                       const OEPropDB::OEKey &k3, double t, unsigned int rmode)
void BondTorsionModify(const OEPropDB::OEKey &k1, const OEPropDB::OEKey &k2,
                       const OEPropDB::OEKey &k3, const OEPropDB::OEKey &k4,
                       double torsion, unsigned int rmode)
```

Modifies the torsion (dihedral) angle specified by either three bonds or four atoms. The amount the angle is modified by is specified by the *torsion* parameter. The portion of the molecule which is moved by this function is specified by the *rmode* parameter. Available values include:

- 0 - the smaller portion of the molecule moves
- 1 - the larger portion of the molecule moves

## 6.59 BondTorsionSet

```
void BondTorsionSet(const OEPropDB::OEKey &k1, const OEPropDB::OEKey &k2,
                    const OEPropDB::OEKey &k3, double t, unsigned int rmode)
void BondTorsionSet(const OEPropDB::OEKey &k1, const OEPropDB::OEKey &k2,
                    const OEPropDB::OEKey &k3, const OEPropDB::OEKey &k4,
                    double torsion, unsigned int rmode)
```

Sets the torsion (dihedral) angle specified by either three bonds or four atoms. The desired angle is specified by the *torsion* parameter. The portion of the molecule which is moved by this function is specified by the *rmode* parameter. Available values include:

- 0 - the smaller portion of the molecule moves
- 1 - the larger portion of the molecule moves

## 6.60 BuilderActiveGet

```
bool BuilderActiveGet()
```

Returns whether or not VIDA is in molecule editing mode.

## 6.61 BuilderActiveSet

```
void BuilderActiveSet(bool)
```

Sets whether or not VIDA is in molecule editing mode.

## 6.62 BuilderAlternateConfModeGet

```
bool BuilderAlternateConfModeGet()
```

Returns whether or not VIDA is in ring alternate conformation exploration mode.

## 6.63 BuilderAlternateConfModeSet

```
void BuilderAlternateConfModeSet(bool)
```

Sets whether or not VIDA is in ring alternate conformation exploration mode.

## 6.64 BuilderAlternateConfSet

```
void BuilderAlternateConfSet(unsigned int)
```

Applies the specified alternate ring conformation to the molecule currently being edited. This is intended for internal use only.

## 6.65 BuilderCancel

```
void BuilderCancel()
```

Cancels all modifications made to the molecule currently being edited and exits editing mode.

## 6.66 BuilderEdit

```
void BuilderEdit(const OEPropDB::OEKey &, bool force3D)
```

Puts VIDA into editing mode for the specified molecule. If the molecule is a 2D molecule, setting the *force3D* parameter to True will ensure that 3D coordinates will be generated, even if the user cancels out of editing mode.

## 6.67 BuilderFinish

```
void BuilderFinish()
```

Commits all modifications made to the molecule currently being edited and exits editing mode.

## 6.68 BuilderFragmentGet

```
std::string BuilderFragmentGet()
```

Returns a hexadecimal encoded OEB string representation of the current fragment being used in the builder component.

## 6.69 BuilderFragmentSet

```
void BuilderFragmentSet(const std::string &oeb)
void BuilderFragmentSet(const std::string &smiles, int)
```

Sets the default fragment to be used in mouse actions in the builder. The first implementation takes a hexadecimal encoded OEB string representation of the molecule fragment (this is primarily intended for internal use). The second implementation takes a SMILES string with an additional index parameter which specifies the index of the atom to be used as the attachment point.

## 6.70 BuilderMinimize

```
void BuilderMinimize()
```

Minimizes the molecule currently being edited using the MMFF forcefield.

## 6.71 BuilderMinimizeScoped

```
void BuilderMinimizeScoped(unsigned int scope = SelectedScope)
```

Minimizes a portion of the molecule currently being edited as determined by the atoms in the specified scope. Valid values for the scope parameter are:

- MarkedScope
- SelectedScope
- ScratchScope

## 6.72 BuilderPropertiesGet

```
std::vector<string> BuilderPropertiesGet()
```

Returns a list of the properties displayed in the Property table of the Builder window.

## 6.73 BuilderPropertyAdd

```
void BuilderPropertyAdd(const std::string &name, const std::string &command, unsigned int flags = Cal
```

Adds a user-defined property calculator to the Properties table of the Builder window. The *name* parameter is the name of the property to be calculated. The *command* parameter is a string containing a Python command to be executed to calculate the property. The return value of the command is displayed in the property table. The *flags* parameter specifies what types of molecules the property will be calculated for, and whether a change to a molecule's coordinates requires the property to be recalculated. Valid values for this parameter are:

- UpdateOnCoordChange
- CalculateForSmallMols
- CalculateForProteins

These values can be OR'ed together to be used in combinations. The property is always recalculated when the chemical graph of the active molecule changes.

To add one of VIDA's default properties (*e.g.*, 'LogP') to the property table, use its name and leave the *command* string empty.

## 6.74 BuilderPropertyRemove

```
void BuilderPropertyRemove(const std::string &propertyToRemove)
```

Removes a property from the Property table of the Builder window.

## 6.75 BuilderPropertySetValue

```
void BuilderPropertySetValue(const std::string &propertyName, const std::string &value)
```

Directly sets a value in the Property table of the Builder window. The value will be overwritten when the property is updated.

## 6.76 BuilderTorsionActivate

```
void BuilderTorsionActivate(const OEPropDB::OEKey &key)
```

Activates an existing torsion monitor, specified by key, so that it can be rotated interactively while in Editing mode. Activating a torsion will temporarily display two torsion measurements, one for each end of the bond. When one of these measurements is selected, the corresponding end of the molecule will be moved when the mouse wheel or up/down arrows are used. If this command is used outside of Editing mode, the double measurements will be displayed but will not be activated.

## 6.77 BuilderTorsionsDeactivate

```
void BuilderTorsionsDeactivate()
```

Returns all active torsion monitors to their inactive state.

## 6.78 BuilderUseMMFF94sSet

```
void BuilderUseMMFF94sSet(bool useMMFF94s)
```

Causes the builder geometry optimizations to use the MMFF94s force field, which provides more nearly planar geometries for delocalized trigonal nitrogens than the MMFF94 force field. Calling this function with *useMMFF94s* = False will cause the builder to revert to using MMFF94.

## 6.79 BuilderUseMMFF94sGet

```
bool BuilderUseMMFF94sGet()
```

Returns True if the builder is currently set to use the MMFF94s force field, or False if the builder is using the MMFF94 force field.

## 6.80 MoleculeAddExplicitHydrogens

```
void MoleculeAddExplicitHydrogens(const OEPropDB::OEKey &k,
                                  bool polarOnly=false)
```

Adds explicit hydrogens to the specified molecule. If the *polarOnly* parameter is specified, only polar hydrogens will be added.

## 6.81 MoleculeAddHydrogens

```
void MoleculeAddHydrogens(const OEPropDB::OEKey &k, bool polarOnly=false)
```

This function adds hydrogens to the molecule associated with the specified key. If the *polarOnly* parameter is set, only polar hydrogens will be added.

## 6.82 MoleculeAddHydrogensScoped

```
void MoleculeAddHydrogensScoped(bool polarOnly=false,
                                unsigned int scope = BestScope)
```

This function adds hydrogens to the molecules int the specified scope. If the *polarOnly* parameter is set, only polar hydrogens will be added.

## 6.83 MoleculeDeleteHydrogens

```
void MoleculeDeleteHydrogens(const OEPropDB::OEKey &k, bool nonpolar=false)
```

Deletes hydrogens off of the specified molecule. If the *nonpolar* parameter is True, it will only delete non-polar hydrogens, leaving the molecule with only polar hydrogens still attached.

## 6.84 MoleculeGenerateCoords

```
void MoleculeGenerateCoords(const OEPropDB::OEKey &k,
                            unsigned int maxconfs=1,
                            bool strict=false)
```

Generates new coordinates for the specified molecule. The maximum number of conformations can be specified using the *maxconfs* parameter. Strict stereo can be turn on using the *strict* parameter.

## 6.85 MoleculeGenerateCoordsFixed

```
void MoleculeGenerateCoordsFixed(const OEPropDB::OEKey &k,
                                 const std::vector<OEPropDB::OEKey> &fixed,
                                 bool match=true, unsigned int maxconfs=1,
                                 bool strict=false)
```

Generates new coordinates for the specified molecule while holding fixed either the specified atoms (if the *match* parameter is True) or all the unspecified atoms (if the *match* parameter is False). The maximum number of conformations can be specified use the *maxconfs* parameter. Strict stereo can be turn on using the *strict* parameter.

## 6.86 MoleculeGenerateCoordsFixedScoped

```
void MoleculeGenerateCoordsFixedScoped(const OEPropDB::OEKey &k,
                                       bool match=true, unsigned int maxconfs=1,
                                       unsigned int scope = SelectedScope,
                                       bool strict)
```

Generates new coordinates for the specified molecule while holding fixed either all the atoms in the specified scope (if the *match* parameter is True) or all the atoms not in specified scope (if the *match* parameter is False). The maximum number of conformations can be specified use the *maxconfs* parameter. Strict stereo can be turn on using the *strict* parameter.

## 6.87 MoleculeNew

```
OEPropDB::OEKey MoleculeNew(const std::string &text, unsigned int listid=0)
```

Creates a new molecule from the specified text representation. Allowed text representations include SMILES as well as IUPAC and common names. A FASTA sequence can also be specified if preceded by an angle bracket character ('>').

Assuming that the molecule can be created from the specified text, it will be added to the specified list (*listid*). If the list ID is zero, a new list will be created.

Returns the key corresponding to this molecule.

## 6.88 MoleculeRotate

```
void MoleculeRotate(const OEPropDB::OEKey &key,
                    int oldx, oldy, int newx, int newy)
```

Rotates a molecule's coordinates corresponding to a mouse move in the 3D display window.

## 6.89 SketcherInputSet

```
void SketcherInputSet(const std::string &smilesString)
void SketcherInputSet(OEMol &molecule)
```

Sets the molecule in the Sketcher from the provided SMILES string or molecule. If an existing 3D molecule is already in build mode, this method will throw an exception.

## 6.90 SketcherLookupFunctionSet

```
void SketcherLookupFunctionSet(const std::string &functionString)
```

Sets a Python function to be called for compound name lookups. The lookup function should accept a string, convert that to either SMILES or a molecule, and call `SketcherInputSet` to put the resulting molecule into the Sketcher. The `functionString` argument should include only the name of the lookup function, without parentheses or arguments, *e.g.*, 'MyLookupFunction'.

CHAPTER

SEVEN

# DATA ANALYSIS FUNCTIONS

The functions detailed in this section provide access to the data analysis and spreadsheet functionality in VIDA. An important distinction should be made in that the "Datatable" functions refer to internal data collections while the "Spreadsheet" functions refer to the actual tabular views of the associated data that actually appear in the Spreadsheet display.

## 7.1 DataAdd

```
void DataAdd(const OEPropDB::OEKey &key, const std::string &str,
             const std::string &data)
```

Adds a piece of data to the object specified by *key*. If a column in the spreadsheet does not already exist for this data type, it will be added.

## 7.2 DataGetDB

```
VFDataBase *DataGetDB()
```

Return the VDDataBase object.

## 7.3 DataGetTable

```
VFDataTable *DataGetTable(const std::string &name, bool throwError=true)
```

Return the list of all internal databases.

## 7.4 DatatableAddColumn

```
void DatatableAddColumn(std::string datatable, std::string columnName,
                        bool genericData=false)
```

Add a column to the specified data table with the specified name. If a column with the specified name already exists, this call will be ignored.

## 7.5 DatatableCommitChanges

```
void DatatableCommitChanges(const std::string &datatable)
```

Commits any outstanding changes to the internal database.

## 7.6 DatatableCurrentGet

```
std::string DatatableCurrentGet()
```

Returns the name of the current data table (e.g. 'Molecules').

## 7.7 DatatableCurrentSet

```
bool DatatableCurrentSet(const std::string &datatable)
```

Sets the current data table.

## 7.8 DatatableData

```
std::string DatatableData(const std::string &datatable, unsigned int row,
                          std::string header)
```

Returns the string representation of the data in the specified data table at the specified row and column.

## 7.9 DatatableDeleteColumn

```
void DatatableDeleteColumn(const std::string &datatable,
                           const std::string &name)
```

Delete the specified column from the specified data table.

## 7.10 DatatableEditableGet

```
bool DatatableEditableGet(const std::string &ss)
```

Returns whether or not the specified data table is editable.

## 7.11 DatatableEditableSet

```
void DatatableEditableSet(const std::string &ss, bool val)
```

Sets whether or not the specified data table is editable.

## 7.12 DatatableFilter

```
void DatatableFilter(const std::string &datatable, const std::string &filter,
                     const std::string &name, bool staticView=true,
                     bool permaFilter=false)
void DatatableFilter(const std::string &datatable, const OEDataFilter &filter,
                     const std::string &name, bool staticView=true,
                     bool permaFilter=false)
```

Creates a new data table view with the specified name from the specified data table using the specified filter. The filter is an arbitrary Python expression that is applied to every row in the source data table to determine whether or not it should be included in the filtered view.

The filter expression may assume that a local variable `ROW` is set to the current row number being evaluated. An example appears below:

```
 int(DatatableData(DatatableCurrent(), ROW, foo)) < 400

The example shows an example expression where the string representation of
the data in the current data table (which may not necessarily be the
specified datatable) at row ``ROW`` and in column ``foo`` is converted to an
integer and compared with the value 400. If this expression is ``True`` for
the row being tested, it will be included in the filtered view, otherwise it
will not be included.
```

## 7.13 DatatableFromList

```
void DatatableFromList(unsigned int listid, const std::string &name)
```

Creates a filtered data table with the specified name with row entries for each object in the specified list. If the specified name already exists, a new name will be used (e.g. *name –> name1*, *name2*, and so on).

## 7.14 DatatableGetColumn

```
std::vector<std::string> DatatableGetColumn(const std::string &datatable,
                                            unsigned int index)
std::vector<std::string> DatatableGetColumn(const std::string &datatable,
                                            const std::string &name)
```

Returns a list of string representations for each entry in the specified column in the specified data table.

## 7.15 DatatableGetCurrentRow

```
int DatatableGetCurrentRow(std::string datatable)
```

Returns the row index for the first active or selected row in the specified data table.

## 7.16 DatatableGetDatatables

```
std::vector<std::string> DatatableGetDatatables()
```

Returns a list of the names of all the available datatables.

## 7.17 DatatableGetImageStreamAtRow

```
bool DatatableGetImageStreamAtRow(const std::string &datatable,
                                  const std::string &filename, int row,
                                  int width=256, int height=256)
```

Write an image file to the file specified by *filename* for the datatable named *datatable* at the row specified by *row*. The size of the image is specified by *width* and *height*.

## 7.18 DatatableGetKeys

```
std::vector<OEKey> DatatableGetKeys(const std::string &datatable)
```

Returns a list of keys for the database specified by *datatable*. The keys are returned in the current datatable sort order and are consistent with the ordering from the call to DatatableGetColumn.

## 7.19 DatatableGetNumRows

```
int DatatableGetNumRows(std::string datatable)
```

Return the number of rows for the specified datatable.

## 7.20 DatatableHeaders

```
std::vector<std::string> DatatableHeaders(const std::string &datatable)
```

Return a list of all the names of all the headers for the specified datatable.

## 7.21 DatatableLingoSimSort

```
bool DatatableLingoSimSort(const std::string &spreadsheet, unsigned int row)
```

Sorts the specified datatable according to Lingo similarity to the molecule contained in the specified row.

## 7.22 DatatableMolNumberFunction

```
double DatatableMolNumberFunction(const std::string &datatable, int row,
                                  const std::string &func)
```

Calculates molecular data that returns a numeric value. *datatable* identifies the datatable containing the molecule and row is the datatable row with the molecule.

The function *func* to compute is one of:

- "mw" Molecular weight.
- "Num Atoms" Number of atoms in the molecule.
- "Num Bonds" number of bonds in the molecule.
- **"Carbon-Hetero ratio" the ratio of carbons to hetero atoms in the molecule.** Returns -1 if there are no carbons.
- "Energy" The molecular energy of the molecule as specified in the input file.
- "Actual Charge" The sum of the partial charges on all atoms as specified in the input file.
- "Formal Charge" The sum of the formal charges on all atoms.
- "Halide Count" The number of halogen atoms in the molecule.
- "Num Carbons" The number of carbon atoms in the molecule.
- "Num Formal Charges" The number of more atoms with a specified formal charge.
- "Num Heavy Atoms" The number of heavy atoms (non hydrogen) in the molecule.
- "Num Hetero Atoms" The number of hetero atoms in the molecule.
- "Num Hydrogens" The number of hydrogen atoms in the molecule.
- "Num Rigid Bonds" The number of rigid bonds in the molecule.
- "Nom Rotatable Bonds" The number of rotatable bonds in the molecule.
- "Num Chiral Atoms" returns the number of chiral atoms in the molecule.
- "Num Chiral Bonds" returns the number of chiral bonds in the molecule.

## 7.23 DatatableMolStringFunction

```
std::string DatatableMolStringFunction(const std::string &datatable, int row,
                                       const std::string &func)
```

Calculates molecular data that returns a string value. *datatable* identifies the datatable containing the molecule and row is the datatable row with the molecule.

The function *func* to compute is one of:

- "molformula" the molecular formula for the molecule.

## 7.24 DatatableNumRows

```
int DatatableNumRows(std::string datatable)
```

Return the number of rows for the datatable named *datatable*.

## 7.25 DatatableSetData

```
void DatatableSetData(const std::string &datatable, unsigned int row,
                      std::string header, std::string value, bool update=true)
```

Set the cell data for the datatable named *datatable* at *row* for the column named *header* to the string value *value*.

If update is True, immediately update the datatable. Set this to False if you are updating a bunch of data and then call `DatatableUpdateContents` on this datatable.

## 7.26 DatatableSetExpression

```
void DatatableSetExpression(const std::string &datatable,
                            const std::string &col, const std::string &expr)
```

A datatable expression defines an arbitrary piece of python code to call that generates data to be displayed in the datatable.

This function creates a new column named *col* for the specified datatable using the function defined by *expr*.

This expression may assume that a local variable *ROW* is assigned to the row currently being evaluated. e.g.

*DatatableData(DatatableCurrent(), ROW, foo)*

returns the string representation of the data in column *foo* for the current datatable which, for this function, is always *datatable*.

## 7.27 DatatableSetRowData

```
void DatatableSetRowData(const std::string &datatable, unsigned int row,
                         std::vector<std::string> headers,
                         std::vector<std::string> rows, bool update=true)
```

Set data in the datatable named *datatable*. *row* is the row index to set. *headers* defines the names of the columns to set and *rows* is the string representation of the data.

**Note**: headers[i] should be the name of the column for row[i].

*The update parameter is no longer used and is kept for backwards compatibility.*

## 7.28 SpreadsheetAddColumn

```
void SpreadsheetAddColumn(std::string spreadsheet, std::string column,
                          bool genericData=false)
```

Add a column to spreadsheet *spreadsheet* with the name *columnName*. If *columnName* already exists in the spreadsheet, it will be ignored.

For the "Atoms" and "Residues" spreadsheets, only certain column names are allowed, and the values in these columns cannot be changed.

For the "Atoms" spreadsheet, the valid column names are:

- x

- y
- z
- Radius
- Element
- Residue_Idx
- Name
- oe_atom_Formal_Charge
- oe_atom_Partial_Charge
- oe_atom_Idx
- oe_atom_Map_Idx
- oe_atom_Isotope
- oe_atom_Hyb
- oe_atom_Implicit_H_Count
- oe_atom_Explicit_H_Count
- oe_atom_Int_Type
- oe_atom_Type
- oe_atom_Aromatic
- oe_atom_Chiral
- oe_atom_In_Ring
- oe_residue_Name
- oe_residue_Occupancy
- oe_residue_BFactor
- oe_residue_Number
- oe_residue_Serial_Number
- oe_residue_Model_Number
- oe_residue_Fragment_Number
- oe_residue_Secondary_Structure
- oe_residue_Alternate_Location
- oe_residue_Chain_ID
- oe_residue_IsHetAtom

The *oe_residue_* and *oe_atom_* prefixes are not displayed in the spreadsheet headers, and are omitted when referring to these columns in other functions, such as SpreadsheetData and SpreadsheetRemoveColumn.

For the "Residues" spreadsheet, the valid column names are:

- Residue
- Average BFactor
- Min Occupancy

- Max Occupancy
- Alt Group
- Num AltConfs

## 7.29 SpreadsheetColumnColorerSet

```
void SpreadsheetColumnColorerSet(const std::string &spreadsheet,
                                 const std::string &column,
                                 const std::string &colorer, double min,
                                 double max)
```

Sets a colorer for the specified column in the specified spreadsheet. Valid values for the *colorer* parameter include:

- "redtoblue"
- "bluetored"
- "rainbow"
- "reverse rainbow"
- "redyellowblue"
- "greytogreen"

## 7.30 SpreadsheetColumnController

```
void SpreadsheetColumnController()
```

Open the spreadsheet column controller dialog. The column controls allows controlling the visibility and other aspects for the current spreadsheets column.

## 7.31 SpreadsheetColumnFontGet

```
std::string SpreadsheetColumnFontGet(const std::string &spreadsheet,
                                     const std::string &column)
```

Returns the font to be used when rendering text in the specified column of the specified spreadsheet.

## 7.32 SpreadsheetColumnFontSet

```
void SpreadsheetColumnFontSet(const std::string &spreadsheet,
                              const std::string &column,
                              const std::string &font)
```

Sets the font to be used when rendering text in the specified column of the specified spreadsheet.

## 7.33 SpreadsheetColumnReadonly

```
bool SpreadsheetColumnReadonly(std::string spreadsheet, unsigned int col)
bool SpreadsheetColumnReadonly(std::string spreadsheet, unsigned int col,
                               bool readonly)
```

Set a spreadsheet column to be read-only.

Without a readOnly value, returns the current readonly state for column *col* in *spreadsheet*.

With a readOnly specified, column *col* in the spreadsheet named *spreadsheet* is set to value set by readOnly.

## 7.34 SpreadsheetColumnSigFigGet

```
int SpreadsheetColumnSigFigGet(const std::string &spreadsheet,
                               const std::string &column)
```

Returns the number of significant figures used to display numerical values data in the specified column.

## 7.35 SpreadsheetColumnSigFigSet

```
void SpreadsheetColumnSigFigSet(const std::string &spreadsheet,
                                const std::string &column, int digits)
```

Sets the number of significant figures used to display numerical values data in the specified column.

## 7.36 SpreadsheetCommitChanges

```
void SpreadsheetCommitChanges(const std::string &spreadsheet)
```

Commit any unsaved changes to the spreadsheet named *spreadsheet*.

Normally, this is done automatically behind the scenes but sometimes is necessary to keep the data in the spreadsheet synchronized with the molecules and data in the repository.

## 7.37 SpreadsheetCopy

```
std::string SpreadsheetCopy(const std::string &spreadsheet)
```

Copies to the clipboard and returns the currently selected set of cells in the specified spreadsheet.

## 7.38 SpreadsheetCreateColumnExpression

```
void SpreadsheetCreateColumnExpression()
```

Open the dialog to create a new column expression for the current spreadsheet. A column expression is a user-defined function that populates a column with arbitrary data.

## 7.39 SpreadsheetCreateFilter

```
void SpreadsheetCreateFilter()
```

Open the dialog to generate a new view of the spreadsheet with all rows filtered by an arbitrary python expression.

## 7.40 SpreadsheetCurrentGet

```
std::string SpreadsheetCurrentGet()
```

Returns the name of the current spreadsheet. e.g. 'Molecules'.

## 7.41 SpreadsheetCurrentSet

```
bool SpreadsheetCurrentSet(const std::string &spreadsheet)
```

Sets the current spreadsheet.

## 7.42 SpreadsheetData

```
std::string SpreadsheetData(const std::string &spreadsheet, unsigned int row,
                            std::string header)
```

Return the string representation of the data in the spreadsheet named *spreadsheet* for the row *row* and the column named *header*.

## 7.43 SpreadsheetDeleteColumn

```
void SpreadsheetDeleteColumn(const std::string &column)
```

Delete the column named *column*. Note that this deletes *column* from all spreadsheets.

## 7.44 SpreadsheetEditableGet

```
bool SpreadsheetEditableGet(const std::string &ss)
```

Returns True if *spreadsheet* is editable.

## 7.45 SpreadsheetEditableSet

```
void SpreadsheetEditableSet(const std::string &ss, bool val)
```

Sets *spreadsheet* to be editable if *val* is True, otherwise sets *spreadsheet* to be read-only.

## 7.46 SpreadsheetFilter

```
void SpreadsheetFilter(const std::string &spreadsheet, std::string filter,
                       std::string name, bool staticView)
void SpreadsheetFilter(const std::string &spreadsheet, const OEDataFilter &p,
                       const std::string &name, bool staticview,
                       bool permaFilter=false)
```

Create a new spreadsheet view from the spreadsheet named *spreadsheet* using the filter defined in *filter*. The new view is named *name*. *filter* is an arbitrary python expression that is applied to every row in the spreadsheet.

This expression may assume that a local variable *ROW* is assigned to the row currently being evaluated. e.g.

SpreadsheetData(SpreadSheetCurrent(), ROW, foo)

returns the string representation of the data in column foo for the current spreadsheet. Note: the current spreadsheet may not be *spreadsheet*.

## 7.47 SpreadsheetFromList

```
void SpreadsheetFromList(unsigned int listid, const std::string &name)
```

Generate a filtered spreadsheet with name *name* from the list *listid*.

If *name* is already in the spreadsheet, it will be renamed name1, name2 and so on.

## 7.48 SpreadsheetGetColumn

```
std::vector<std::string> SpreadsheetGetColumn(const std::string &spreadsheet,
                                              unsigned int index)
std::vector<std::string> SpreadsheetGetColumn(const std::string &spreadsheet,
                                              const std::string &name)
```

Returns string representations of the entire column for the spreadsheet named *spreadsheet* for column *name* or, alternatively, for column index by *index*.

## 7.49 SpreadsheetGetCurrentRow

```
int SpreadsheetGetCurrentRow(std::string spreadsheet)
```

Returns the row index for the first active or selected row for the spreadsheet named *spreadsheet*.

## 7.50 SpreadsheetGetIDForRow

```
unsigned int SpreadsheetGetIDForRow(std::string spreadsheet, unsigned int row)
```

Returns the repository id for the first active or selected row for the spreadsheet named *spreadsheet*.

## 7.51 SpreadsheetGetImageStreamAtRow

```
bool SpreadsheetGetImageStreamAtRow(const std::string &spreadsheet,
                                    const std::string &filename, int row,
                                    int width=256, int height=256)
```

Write an image file to the file specified by *filename* for the spreadsheet named *spreadsheet* at the row specified by *row*. The size of the image is specified by *width* and *height*.

## 7.52 SpreadsheetGetKeyForRow

```
OEPropDB::OEKey SpreadsheetGetKeyForRow(std::string spreadsheet,
                                        unsigned int row)
```

Returns the OEKey value for the first active or selected row for the spreadsheet named *spreadsheet*.

## 7.53 SpreadsheetGetNumRows

```
int SpreadsheetGetNumRows(std::string spreadsheet)
```

Return the number of rows for the spreadsheet named *spreadsheet*.

## 7.54 SpreadsheetGetRowForKey

```
unsigned int SpreadsheetGetRowForKey(std::string spreadsheet, OEPropDB::OEKey)
```

Returns the row that is generated from OEKey *key*. Returns "4294967295L" if *key* is not in the spreadsheet.

## 7.55 SpreadsheetGetSpreadsheets

```
std::vector<std::string> SpreadsheetGetSpreadsheets()
```

Return a list of the names for all the currently available spreadsheets.

## 7.56 SpreadsheetHeaders

```
std::vector<std::string> SpreadsheetHeaders(const std::string &spreadsheet)
```

Return a list of all the names of all the headers for the spreadsheet named *spreadsheet*.

## 7.57 SpreadsheetHideColumn

```
void SpreadsheetHideColumn(const std::string &spreadsheet,
                           const std::string &name)
```

Hide the column named *name* for the spreadsheet named *spreadsheet*.

## 7.58 SpreadsheetHideTab

```
bool SpreadsheetHideTab(std::string name)
```

Hide the named spreadsheet from the spreadsheet view.

## 7.59 SpreadsheetImport

```
int SpreadsheetImport(const VFSpreadSplitter &splitter,
                      const std::string &filename,
                      unsigned int matchBy=ImportSpreadsheet::ImportByOrder,
                      const std::string &matchList="",
                      unsigned int importBy=ImportSpreadsheet::ImportAsIs,
                      const std::string &importColumnOrFunction="")
```

Imports the specified file into the spreadsheet.

## 7.60 SpreadsheetLingoSimSort

```
void SpreadsheetLingoSimSort(const std::string &spreadsheet, unsigned int row)
```

Sorts the specified spreadsheet according to Lingo similarity to the molecule in the specified row.

## 7.61 SpreadsheetLoadFilter

```
void SpreadsheetLoadFilter(const std::string &spreadsheet,
                           const OEDataFilter &filter)
void SpreadsheetLoadFilter(const std::string &spreadsheet,
                           const OEDataFilter &filter, bool wasStatic,
                           bool isStatic)
```

Creates a new spreadsheet using the specified filter.

## 7.62 SpreadsheetMolNumberFunction

```
double SpreadsheetMolNumberFunction(const std::string &spreadsheet, int row,
                                    const std::string &func)
```

Calculates molecular data that returns a numeric value.

*spreadsheet* identifies the spreadsheet containing the molecule and row is the spreadsheet row with the molecule.

The function *func* to compute is one of:

- "mw" Molecular weight.
- "Num Atoms" Number of atoms in the molecule.
- "Num Bonds" number of bonds in the molecule.
- **"Carbon-Hetero ratio" the ratio of carbons to hetero atoms in the molecule.** Returns -1 if there are no carbons.

- "Energy" The molecular energy of the molecule as specified in the input file.
- "Actual Charge" The sum of the partial charges on all atoms as specified in the input file.
- "Formal Charge" The sum of the formal charges on all atoms.
- "Halide Count" The number of halogen atoms in the molecule.
- "Num Carbons" The number of carbon atoms in the molecule.
- "Num Formal Charges" The number of more atoms with a specified formal charge.
- "Num Heavy Atoms" The number of heavy atoms (non hydrogen) in the molecule.
- "Num Hetero Atoms" The number of hetero atoms in the molecule.
- "Num Hydrogens" The number of hydrogen atoms in the molecule.
- "Num Rigid Bonds" The number of rigid bonds in the molecule.
- "Nom Rotatable Bonds" The number of rotatable bonds in the molecule.
- "Num Chiral Atoms" returns the number of chiral atoms in the molecule.
- "Num Chiral Bonds" returns the number of chiral bonds in the molecule.

## 7.63 SpreadsheetMolStringFunction

```
std::string SpreadsheetMolStringFunction(const std::string &spreadsheet,
                                         int row, const std::string &func)
```

Calculate molecular data that returns a string value.

*spreadsheet* identifies the spreadsheet containing the molecule and row is the spreadsheet row with the molecule.

The function *func* to compute is one of:

- 'molformula' the molecular formula for the molecule.

## 7.64 SpreadsheetMoveColumn

```
void SpreadsheetMoveColumn(const std::string &spreadsheet,
                           const std::string &columnName, int position)
```

Moves the specified column to the specified position.

## 7.65 SpreadsheetNumRows

```
int SpreadsheetNumRows(std::string spreadsheet)
```

Return the number of rows for the spreadsheet named *spreadsheet*.

## 7.66 SpreadsheetPromptColumnExpression

```
std::string SpreadsheetPromptColumnExpression()
```

This function opens a dialog and returns the expression to evaluate to add the specified column expression.

## 7.67 SpreadsheetPromptExport

```
std::string SpreadsheetPromptExport(const std::string &filename)
```

This function opens a dialog and returns the expression to export the specified spreadsheet and columns.

## 7.68 SpreadsheetPromptFilter

```
std::string SpreadsheetPromptFilter()
```

This function opens a dialog and returns the expression to evaluate to filter the specified spreadsheet.

## 7.69 SpreadsheetPromptFormat

```
std::string SpreadsheetPromptFormat()
```

This function opens a dialog which allows the user to specify the formatting of the spreadsheet.

## 7.70 SpreadsheetPromptGraphemeOpts

```
std::string SpreadsheetPromptGraphemeOpts()
```

This function opens a dialog which allows the user to specify how the 2D depiction is rendered using OpenEye's **Grapheme** toolkit to apply a property map of computed properties or even user defined properties which are tagged to the atoms using generic data.

## 7.71 SpreadsheetPromptImport

```
std::string SpreadsheetPromptImport(const std::string &filename)
```

This function opens a dialog and returns the expression to import spreadsheet *filename*.

## 7.72 SpreadsheetPromptSort

```
std::string SpreadsheetPromptSort()
```

This function opens a dialog and returns the expression to evaluate to sort the specified spreadsheet and columns.

## 7.73 SpreadsheetRemoveTab

```
bool SpreadsheetRemoveTab(std::string name)
```

Remove the spreadsheet with name *name*. The base "Molecules" spreadsheet cannot be removed.

## 7.74 SpreadsheetRowHeightGet

```
int SpreadsheetRowHeightGet(const std::string &spreadsheet)
```

Returns the row height for the specified spreadsheet.

## 7.75 SpreadsheetRowHeightSet

```
void SpreadsheetRowHeightSet(const std::string &spreadsheet, int height)
```

Sets the default row height for the specified spreadsheet.

## 7.76 SpreadsheetSetData

```
void SpreadsheetSetData(const std::string &spreadsheet, unsigned int row,
                        std::string header, std::string value, bool update=true)
```

Set the cell data for the spreadsheet named *spreadsheet* at *row* for the column named *header* to the string value *value*.

If update is true, immediately update the spreadsheet. Set this to false if you are updating a bunch of data and then call `SpreadsheetUpdateContents` on the spreadsheet.

## 7.77 SpreadsheetSetExpression

```
void SpreadsheetSetExpression(const std::string &spreadsheet,
                              const std::string &col, const std::string &expr)
```

A spreadsheet expression defines an arbitrary piece of python code to call that generates data to be displayed in the spreadsheet.

SpreadsheetSetExpression creates a new column named *col* for the spreadsheet named *spreadsheet* using the function defined by *expr*.

This expression may assume that a local variable *ROW* is assigned to the row currently being evaluated. e.g.

SpreadsheetData(SpreadSheetCurrent(), ROW, foo)

returns the string representation of the data in column *foo* for the current spreadsheet which, for this function, is always *spreadsheet*.

## 7.78 SpreadsheetSetRowData

```
void SpreadsheetSetRowData(const std::string &spreadsheet, unsigned int row,
                           std::vector<std::string> headers,
                           std::vector<std::string> rows, bool update=true)
```

Set data in the spreadsheet named *spreadsheet*. *row* is the row index to set. *headers* defines the names of the columns to set and *rows* is the string representation of the data.

**Note**: headers[i] should be the name of the column for row[i].

If update is true, immediately update the spreadsheet. Set this to false if you are updating a bunch of data and then call SpreadsheetUpdateContents(ss).

## 7.79 SpreadsheetShowAllColumns

```
void SpreadsheetShowAllColumns(const std::string &spreadsheet)
```

Make all columns visible for the spreadsheet named *spreadsheet*.

## 7.80 SpreadsheetShowAllTabs

```
void SpreadsheetShowAllTabs()
```

Make all spreadsheets visible.

## 7.81 SpreadsheetShowColumn

```
void SpreadsheetShowColumn(const std::string &spreadsheet, int section,
                           bool visible=true)
void SpreadsheetShowColumn(const std::string &spreadsheet,
                           const std::string &name, bool visible=true)
```

Make column *name* visible for the spreadsheet named *spreadsheet*.

## 7.82 SpreadsheetShowStats

```
void SpreadsheetShowStats(const std::string &spreadsheet, bool show)
```

If *show* is true, show the column statistics for the spreadsheet named *spreadsheet*. Otherwise, hide the statistics.

## 7.83 SpreadsheetShowStatsGet

```
bool SpreadsheetShowStatsGet(const std::string &spreadsheet)
```

Returns True if *spreadsheets* has the statistics window shown, False otherwise.

## 7.84 SpreadsheetShowStatsSet

```
void SpreadsheetShowStatsSet(const std::string &spreadsheet, bool show)
```

If *show* is True, the statistics window for *spreadsheet* is shown. Otherwise it is hidden.

## 7.85 SpreadsheetShowTab

```
bool SpreadsheetShowTab(std::string name)
```

Ensure that *spreadsheet* is shown as a spreadsheet selection in the spreadsheet window.

## 7.86 SpreadsheetSort

```
void SpreadsheetSort(const std::string &spreadsheet,
                     const std::vector<std::string> &columns,
                     const std::vector<int> &directions,
                     bool moveToFirst=true)
```

Sort *spreadsheet* by the specified columns and directions. If a direction is 1 then the column is ascending, if a direction is 2 the column is descending.

Example:

SpreadsheetSort( “Molecules”, [”target”, “IC50”], [1,2] )

target | IC50 cox2 | 1.60 cox2 | 1.40 cox1 | 1.80 cox1 | 1.67

If *moveToFirst* is True then the sorted columns will be placed first in the spreadsheet.

This would sort the molecules by target ascending and then by IC50 descending

CHAPTER

# EIGHT

# DEPRECATED FUNCTIONS

The functions detailed in this section have been deprecated. They are still available for use, but are scheduled to be removed from the API in a future version. Most of these functions have been deprecated in order to create a more consistently named API, but some have been deprecated as the functionality they provided is no longer available, obsolete, or unnecessary. If an alternate function is available, it will be referenced in the documentation below.

## 8.1 ContourCurrentGridGet

```
unsigned int ContourCurrentGridGet()
```

This function has been deprecated.

## 8.2 ContourCurrentGridSet

```
unsigned int ContourCurrentGridSet(unsigned int id)
```

This function has been deprecated.

## 8.3 CreateAngleMonitor

```
unsigned int CreateAngleMonitor(const OEPropDB::OEKey &k1,
                                const OEPropDB::OEKey &k2,
                                const OEPropDB::OEKey &k3)
```

This function has been deprecated. Please use `MonitorAngleCreate` instead.

## 8.4 CreateDistanceMonitor

```
unsigned int CreateDistanceMonitor(const OEPropDB::OEKey &k1,
                                   const OEPropDB::OEKey &k2)
```

This function has been deprecated. Please use `MonitorDistanceCreate` instead.

## 8.5 CreateSphereMonitor

```
unsigned int CreateSphereMonitor(const OEPropDB::OEKey &k,
                                 const std::string &name,
                                 const OESystem::OEColor &c, float rad=0.0f)
unsigned int CreateSphereMonitor(const OEPropDB::OEKey &k,
                                 const std::string &name, float x, float y,
                                 float z, float rad, const OESystem::OEColor &c)
```

This function has been deprecated. Please use `MonitorSphereCreate` instead.

## 8.6 CreateTorsionMonitor

```
unsigned int CreateTorsionMonitor(const OEPropDB::OEKey &k1,
                                  const OEPropDB::OEKey &k2,
                                  const OEPropDB::OEKey &k3,
                                  const OEPropDB::OEKey &k4)
```

This function has been deprecated. Please use `MonitorTorsionCreate` instead.

## 8.7 DatatableCurrent

```
std::string DatatableCurrent()
```

This function has been deprecated. Please use `DatatableCurrentGet` instead.

## 8.8 DeleteAngleMonitor

```
void DeleteAngleMonitor(const OEPropDB::OEKey &k1, const OEPropDB::OEKey &k2,
                        const OEPropDB::OEKey &k3)
```

This function has been deprecated. Please use `MonitorAngleDelete` instead.

## 8.9 DeleteDistanceMonitor

```
void DeleteDistanceMonitor(const OEPropDB::OEKey &k1,
                           const OEPropDB::OEKey &k2)
```

This function has been deprecated. Please use `MonitorDistanceDelete` instead.

## 8.10 DeleteTorsionMonitor

```
void DeleteTorsionMonitor(const OEPropDB::OEKey &k1, const OEPropDB::OEKey &k2,
                          const OEPropDB::OEKey &k3, const OEPropDB::OEKey &k4)
```

This function has been deprecated. Please use `MonitorTorsionDelete` instead.

## 8.11 DeleteVisibleMonitors

```
void DeleteVisibleMonitors()
```

This function has been deprecated. Please use `MonitorsVisibleDelete` instead.

## 8.12 ExistsAngleMonitor

```
unsigned int ExistsAngleMonitor(const OEPropDB::OEKey &k1,
                                const OEPropDB::OEKey &k2,
                                const OEPropDB::OEKey &k3)
```

This function has been deprecated. Please use `MonitorAngleExists` instead.

## 8.13 ExistsDistanceMonitor

```
unsigned int ExistsDistanceMonitor(const OEPropDB::OEKey &k1,
                                   const OEPropDB::OEKey &k2)
```

This function has been deprecated. Please use `MonitorDistanceExists` instead.

## 8.14 ExistsTorsionMonitor

```
unsigned int ExistsTorsionMonitor(const OEPropDB::OEKey &k1,
                                  const OEPropDB::OEKey &k2,
                                  const OEPropDB::OEKey &k3,
                                  const OEPropDB::OEKey &k4)
```

This function has been deprecated. Please use `MonitorTorsionExists` instead.

## 8.15 GetAtomsByScope

```
std::vector<OEPropDB::OEKey> GetAtomsByScope(unsigned int scope)
```

This function has been deprecated. Please use `GetAtomsScoped` instead.

## 8.16 GetBondsByScope

```
std::vector<OEPropDB::OEKey> GetBondsByScope(unsigned int scope)
```

This function has been deprecated. Please use `GetBondsScoped` instead.

## 8.17 GetContoursByScope

```
std::vector<OEPropDB::OEKey> GetContoursByScope(unsigned int scope)
```

This function has been deprecated. Please use GetContoursScoped instead.

## 8.18 GetGridsByScope

```
std::vector<OEPropDB::OEKey> GetGridsByScope(unsigned int scope)
```

This function has been deprecated. Please use GetGridsScoped instead.

## 8.19 GetKey

```
OEPropDB::OEKey GetKey(const OESystem::OEBase &base)
OEPropDB::OEKey GetKey(const OESystem::OEBase &base,
                       const OEPropDB::OEKey &parentKey)
```

This function has been deprecated. Please use KeyGet instead.

## 8.20 GetKeyType

```
std::string GetKeyType(const OEPropDB::OEKey &key)
```

This function has been deprecated. Please use KeyTypeGet instead.

## 8.21 GetKeysByScope

```
std::vector<OEPropDB::OEKey> GetKeysByScope(unsigned int scope,
                                            unsigned int type)
```

This function has been deprecated. Please use GetScoped instead.

## 8.22 GetKeysForID

```
std::vector<OEPropDB::OEKey> GetKeysForID(unsigned int id)
```

This function has been deprecated. Please use KeysGet instead.

## 8.23 GetMoleculesByScope

```
std::vector<OEPropDB::OEKey> GetMoleculesByScope(unsigned int scope)
```

This function has been deprecated. Please use GetMoleculesScoped instead.

## 8.24 GetName

```
std::string GetName(const OEPropDB::OEKey &key)
```

This function has been deprecated. Please use `NameGet` instead.

## 8.25 GetPropertyType

```
unsigned int GetPropertyType(const std::string &type)
```

This function has been deprecated. Please use `PropertyTypeGet` instead.

## 8.26 GetSurfacesByScope

```
std::vector<OEPropDB::OEKey> GetSurfacesByScope(unsigned int scope)
```

This function has been deprecated. Please use `GetSurfacesScoped` instead.

## 8.27 GotoStylePage

```
void GotoStylePage(const std::string &page)
```

This function has been deprecated.

## 8.28 InitializeObject

```
void InitializeObject(unsigned int id)
```

This function has been deprecated. Please use `Initialize` instead.

## 8.29 InterpreterInteractiveGet

```
bool InterpreterInteractiveGet()
```

This function has been deprecated.

## 8.30 InterpreterInteractiveSet

```
void InterpreterInteractiveSet(bool interactive)
```

This function has been deprecated.

## 8.31 MarkObjectsByScope

```
bool MarkObjectsByScope(unsigned int scope)
```

This function has been deprecated. Please use MarkScoped instead.

## 8.32 MarkState

```
int MarkState()
```

This function has been deprecated. Please use StateMark instead.

## 8.33 MenuAddLabel

```
std::string MenuAddLabel(const std::string &menu, const std::string &name,
                         bool last)
```

This function has been deprecated.

## 8.34 MenuUseTearoffsGet

```
bool MenuUseTearoffsGet()
```

This function has been deprecated.

## 8.35 MenuUseTearoffsSet

```
bool MenuUseTearoffsSet(bool b)
```

This function has been deprecated.

## 8.36 OEGetKey

```
OEPropDB::OEKey OEGetKey(const OESystem::OEBase &b)
OEPropDB::OEKey OEGetKey(const OESystem::OEBase &b, const OEPropDB::OEKey &k)
```

This function has been deprecated. Please use KeyGet instead.

## 8.37 OEKeyToID

```
unsigned int OEKeyToID(const OEPropDB::OEKey &k)
```

This function has been deprecated. Please use KeyIDGet instead.

## 8.38 OEKeyToLabelString

```
std::string OEKeyToLabelString(const OEPropDB::OEKey &k, bool full=false,
                               bool coords=true)
```

This function has been deprecated. Please use `LabelGet` instead.

## 8.39 OEKeyToParentID

```
unsigned int OEKeyToParentID(const OEPropDB::OEKey &k)
```

This function has been deprecated. Please use `KeyParentIDGet` instead.

## 8.40 OEKeyToSourceID

```
unsigned int OEKeyToSourceID(const OEPropDB::OEKey &k)
```

This function has been deprecated. Please use `KeySourceIDGet` instead.

## 8.41 OEPyClearActive

```
void OEPyClearActive()
```

This function has been deprecated. Please use `ClearActive` instead.

## 8.42 OEPyClearLocked

```
void OEPyClearLocked()
```

This function has been deprecated. Please use `ClearLocked` instead.

## 8.43 OEPyClearMarked

```
void OEPyClearMarked()
```

This function has been deprecated. Please use `ClearMarked` instead.

## 8.44 OEPyClearSelected

```
void OEPyClearSelected()
```

This function has been deprecated. Please use `ClearSelected` instead.

## 8.45 OEPyClearVisible

```
void OEPyClearVisible()
```

This function has been deprecated. Please use `ClearVisible` instead.

## 8.46 OEPyGetActive

```
OEPropDB::OEKey OEPyGetActive()
```

This function has been deprecated. Please use `ActiveKey` instead.

## 8.47 OEPyGetIDForKey

```
unsigned int OEPyGetIDForKey(const OEPropDB::OEKey &k)
```

This function has been deprecated. Please use `KeyIDGet` instead.

## 8.48 OEPyGetSelectedAtoms

```
std::vector<OEPropDB::OEKey> OEPyGetSelectedAtoms()
```

This function has been deprecated. Please use `GetSelectedAtoms` instead.

## 8.49 OEPyHide

```
bool OEPyHide(unsigned int scope, bool value)
```

This function has been deprecated. Please use `HideScoped` instead.

## 8.50 OEPyHideNone

```
bool OEPyHideNone(unsigned int scope)
```

This function has been deprecated. Please use `HideNonScoped` instead.

## 8.51 OEPyHideOthers

```
bool OEPyHideOthers(const std::vector<OEPropDB::OEKey> &keys, bool hide)
```

This function has been deprecated. Please use `HideOthers` instead.

## 8.52 OEPyIsActive

```
bool OEPyIsActive(const OEPropDB::OEKey &k)
```

This function has been deprecated. Please use `IsActive` instead.

## 8.53 OEPyIsLocked

```
bool OEPyIsLocked(const OEPropDB::OEKey &k)
```

This function has been deprecated. Please use `IsLocked` instead.

## 8.54 OEPyIsMarked

```
bool OEPyIsMarked(const OEPropDB::OEKey &k)
```

This function has been deprecated. Please use `IsMarked` instead.

## 8.55 OEPyIsSelected

```
bool OEPyIsSelected(const OEPropDB::OEKey &k)
```

This function has been deprecated. Please use `IsSelected` instead.

## 8.56 OEPyIsTrulyVisible

```
bool OEPyIsTrulyVisible(const OEPropDB::OEKey &k)
```

This function has been deprecated. Please use `IsTrulyVisible` instead.

## 8.57 OEPyIsVisible

```
bool OEPyIsVisible(const OEPropDB::OEKey &k)
```

This function has been deprecated. Please use `IsVisible` instead.

## 8.58 OEPySetActive

```
bool OEPySetActive(const OEPropDB::OEKey &k)
```

This function has been deprecated. Please use `Active` instead.

## 8.59 OEPySetLocked

```
bool OEPySetLocked(const OEPropDB::OEKey &k, bool state)
bool OEPySetLocked(const std::vector<OEPropDB::OEKey> &keys, bool state)
```

This function has been deprecated. Please use `Lock` instead.

## 8.60 OEPySetMarked

```
bool OEPySetMarked(const OEPropDB::OEKey &k, bool state)
bool OEPySetMarked(const std::vector<OEPropDB::OEKey> &keys, bool state)
```

This function has been deprecated. Please use `Mark` instead.

## 8.61 OEPySetSelected

```
bool OEPySetSelected(const OEPropDB::OEKey &k, bool state)
bool OEPySetSelected(const std::vector<OEPropDB::OEKey> &keys, bool state)
```

This function has been deprecated. Please use `Select` instead.

## 8.62 OEPySetVisible

```
bool OEPySetVisible(const OEPropDB::OEKey &k, bool state)
bool OEPySetVisible(const std::vector<OEPropDB::OEKey> &keys, bool state)
```

This function has been deprecated. Please use `Visible` instead.

## 8.63 ObjectNameGet

```
std::string ObjectNameGet(unsigned int id)
std::string ObjectNameGet(const OEPropDB::OEKey &k)
```

This function has been deprecated. Please use `NameGet` instead.

## 8.64 ObjectNameSet

```
void ObjectNameSet(unsigned int id, const std::string &name)
void ObjectNameSet(const OEPropDB::OEKey &k, const std::string &name)
```

This function has been deprecated. Please use `NameSet` instead.

## 8.65 ObservableOEKey

```
OEGUI::OEObservableBase<OEPropDB::OEKey> &
  ObservableOEKey(const std::string &name, bool create=false)
```

This function has been deprecated. Please use `ObservableKey` instead.

## 8.66 PopState

```
int PopState(bool load=true)
```

This function has been deprecated. Please use `StatePop` instead.

## 8.67 PopupAddLabel

```
std::string PopupAddLabel(const std::string &label, bool once=false)
```

This function has been deprecated.

## 8.68 ResponseVectMulti

```
void ResponseVectMulti(const std::string &s,
                       const std::vector<OEInterpreter::OEMultiTypeVar> &var,
                       bool b=false)
```

This function has been deprecated. Please use `PromptResponseVectMulti` instead.

## 8.69 SDataGet

```
std::string SDataGet(unsigned int id, const std::string &tag)
std::string SDataGet(const OEPropDB::OEKey &k, const std::string &tag)
```

This function has been deprecated. Please use `SDDataGet` instead.

## 8.70 SDataHas

```
bool SDataHas(unsigned int id, const std::string &tag)
bool SDataHas(const OEPropDB::OEKey &k, const std::string &tag)
```

This function has been deprecated. Please use `SDDataHas` instead.

## 8.71 SDataSet

```
void SDataSet(unsigned int id, const std::string &tag, const std::string &v)
void SDataSet(const OEPropDB::OEKey &k, const std::string &tag,
              const std::string &v)
```

This function has been deprecated. Please use SDDataSet instead.

## 8.72 ScratchList

```
std::vector<OEPropDB::OEKey> ScratchList()
```

This function has been deprecated. Please use ScratchGet instead.

## 8.73 SpreadsheetCurrent

```
std::string SpreadsheetCurrent()
```

This function has been deprecated, please use SpreadsheetCurrentGet instead.

## 8.74 SpreadsheetDisableUpdates

```
void SpreadsheetDisableUpdates(const std::string &spreadsheet)
```

This function has been deprecated.

## 8.75 SpreadsheetEnableUpdates

```
void SpreadsheetEnableUpdates(const std::string &spreadsheet)
```

This function has been deprecated.

## 8.76 SpreadsheetMarkHighlighted

```
void SpreadsheetMarkHighlighted(const std::string &spreadsheet, bool marked)
```

This function has been deprecated.

## 8.77 SpreadsheetUpdateContents

```
void SpreadsheetUpdateContents(const std::string &spreadsheet)
```

This function has been deprecated.

## 8.78 SurfacePotentialColorAuto

```
void SurfacePotentialColorAuto()
```

This function has been deprecated.

## 8.79 SurfacePotentialColorValuesSet

```
void SurfacePotentialColorValuesSet(float min, float mid, float max)
```

This function has been deprecated.

## 8.80 VFCopyFile

```
bool VFCopyFile(const std::string &src, const std::string &dst)
```

This function has been deprecated.

## 8.81 VFGetMol

```
bool VFGetMol(OEChem::OEMCMolBase &m, unsigned int id)
bool VFGetMol(OEChem::OEMCMolBase &m, const OEPropDB::OEKey &key)
```

This function has been deprecated. Please use `MoleculeGet` instead.

## A

## B

## C

## D

## E



## F



## G



## H

## I



## J



## K



## L

## M

## N



## O



## P

## Q



## R



## S

## T



## U



## V

## W



## X