
# SPRUCE

***Release 1.0.0.2***

**OpenEye Scientific Software, Inc.**

December 11, 2019

CONTENTS

CHAPTER

ONE

# INTRODUCTION

## 1.1 Overview

The **SPRUCE** application provides the facilities to prepare biomolecules (i.e. proteins and nucleic acids) for modeling tasks.

## 1.2 Applications

The **SPRUCE** distribution comprises 3 applications:

*SPRUCE*

The *SPRUCE* program is used for biomolecule preparation, from experimental or modeled structures. The process involves, 1) expansion of the asymmetric unit to the biological unit (if structure is from an X-ray crystallography experiment and it it is necessary), 2) enumeration (default) or collapse of alternate locations, 3) building missing pieces e.g. partial sidechains and capping chain breaks, 4) placement and optimization of hydrogen atoms including tautomer enumeration of ligands and cofactors, as well as evaluation of those tautomer states in the biomolecule structure.

*Superposition*

The *Superposition* program is used to superpose biomolecule structures onto a reference structure. Several different superposition methods are included, that are either based on sequence or secondary structure.

*EnumSites*

The *EnumSites* program can be used to enumerate sites from prior bio-design units produced by *SPRUCE*. This is particularly useful when enumerating potentially allosteric apo sites, that where not known when the structure was prepared the first time - enumeration ensures that the structure is not re-prepared but just focused into a site of interest.

*GetStructure*

The *GetStructure* utility program can be used to download biomolecular structure files (PDB, mmCIF, mtz) for downstream prepping with *SPRUCE*.

## 1.3 Theory

**Spruce TK** is used to process PDB or mmCIF files containing the structures, resulting from X-ray crystallography, Nuclear Magnetic Resonance (NMR) spectroscopy, or electron microscopy (EM) experiments. into molecules usable for molecular modeling. Due to the nature of the experiments and data, some processing is required before use in modeling. SpruceTk provides an end-to-end preparation workflow using the `OEMakeDesignUnits` API. Options

(see `OEMakeDesignUnitOptions` and associated options classes) are available to control for the desired behavior. Metadata (see `OEStructureMetadata`) can be supplied to infuse experimental data, like the proper variant sequence and also fix common problems reading from PDB files (e.g. bond order perception). Spruce has a built in Chemical Components Dictionary matching 3-letter residue codes to SMILES, derived from the corresponding one at the RCSB, but curated to fix improper SMILES. The workflow automatically runs through the below steps to produce `OEDesignUnit` objects:

1. Biological unit extraction (see `OEExtractBioUnits`)
2. If structure is from X-ray, it will also generate packing residues for visualization of crystal contacts
3. Alternate location assignment or enumeration (see the `OEAlternateLocationOption` namespace)
4. Splits the system into components, e.g. identifying ligands, cofactors, excipients.
5. Building missing side-chains and capping chain breaks (see `OEBuildSidechains` and `OECapTermini`)
6. Hydrogen placement and optimization, including searching ligand tautomer states (see `OEProtonateDesignUnit`)
7. Assignment of partial charges to the entire system (see `OEDesignUnitCharges`)
8. Superposition to a common reference frame (see `OEStructuralSuperposition` and `OESecondaryStructureSuperposition`)
9. An `OEInteractionHintContainer` encoded for visualization and more
10. An estimate of the model quality using Iridium [Warren-2012] stored in `OEIridiumData`

---

**Note:** Multiple design units are often produced. This results from multiple biological units generated during step 1, and further expanding that set when enumerating the alternate locations in step 2. The Iridium categorization (see *section_sprucetk_iridium_categorization*) should be helpful in choosing which (if not all) of the produced units to use in downstream modeling tasks. We find that the Iridium categories are also helpful for managing modeling expectations, having a quality measure of the input data for modeling.

---

### 1.3.1 Definitions

1. **Biological Unit (BU)** - the biologically relevant unit for a protein to use in modeling. For example, in Trypsin the BU is a monomer, in HIV Protease the BU is a homo-dimer. See *Biological Unit* below.
2. **Asymmetric Unit (ASU)** - the contents of a PDB or mmCIF file from X-ray contain an asymmetric unit as the output of the experiment. This is sometimes equivalent to the BU, but often requires manipulation to create a correct BU.
3. **Design Unit (DU)** - the result of preparation in spruce is a collection of molecules from a single BU, extracted and ready to use for modeling tasks. See *Design Unit* below for more details.
4. **Alternate locations (alt locs)** - X-ray experiments can often contain results with atoms occupying more than one location. Crystallographers denote this with a measure of the amount of time an atom occupies a given set of coordinates. A well resolved atom will have an occupancy of 1.0, meaning that 100% of the time it is in that location. Sometimes, the atom exists in two positions, alternate locations, and these appear in the input file with single letter designations and with occupancy < 1.0.

### 1.3.2 Biological Unit

In short, the biological unit (BU) is an object that contains the biologically relevant parts of an ASU, which have not been split into various molecular components and are not yet prepped for modeling. For a more detailed explanation of BUs, refer to Introduction to Biological Assemblies and the PDB Archive hosted at the RCSB.

A BU can be constructed from a single PDB from the PDB's own header remarks, or by using a sequence alignment to an input reference protein. To extract a BU or set of BUs from a PDB, one should use the helper function in **Spruce TK** called `OEExtractBioUnits`. The following example shows how to extract BUs from the PDB remarks:

Using the same function, one can also extract BUs from a PDB using an input reference protein. The following example shows how to use `OEExtractBioUnits` to extract BUs from a given reference:

### 1.3.3 Design Unit

The design unit (DU) is an object that contains the extracted and prepared parts of a single BU, ready for modeling. The parts include:

1. Protein
2. Ligand (not always, an apo DU will not contain a ligand)
3. Site residues
4. Packing residues (if any exist near the site)
5. Excipients (if any exist near the site)

### 1.3.4 Protein Superposition

A protein can be structurally superimposed on to a reference protein structure using the **OESpruce TK**. Proteins can be superimposed with either atomic coordinates in the `OEStructuralSuperposition` class, or with secondary structure elements using the `OESecondaryStructureSuperposition` class.

The `OEStructuralSuperposition` class can superimpose proteins using any of the following four methods:

1. The global method (default for `OEStructuralSuperposition`). This method uses all matching heavy atoms from the reference and fit proteins as the region for performing the superposition calculation (see `OESuperpositionType::Global`).
2. The **D**ifference **D**istance **M**atrix method. This method calculates the pairwise distance matrix of C-alpha atoms for both the reference and fit proteins, then takes the difference of these two matrices to find the difference distance matrix (DDM). The longest contiguous region of the resulting difference distance matrix (DDM) is used for the structural superposition calculation (see `OESuperpositionType::DDM`).
3. The weighted-DDM method. This method uses the DDM matrix, and calculates Gaussian weighting factors for all matching C-alpha atoms. These Gaussian weights are used as a weighting function in the superposition calculation (see `OESuperpositionType::Weighted`).
4. The site residue method. This method uses a set of site residues (given as a set of unique residue strings) as a constraint on the protein superposition. Site residues must be set with the `SetSiteResidues` member function of the `OESuperpositionOptions` class (see `OESuperpositionType::Site`).

The `OESecondaryStructureSuperposition` class can superimpose proteins using the following method:

1. The secondary structure superposition method. This method takes two proteins and performs a structural superposition on them based on the shape overlap of the secondary structure elements of the two proteins.

---

**Note:** All structural superposition methods in the `OEStructuralSuperposition` class have a corresponding score from the sequence alignment that was used to find the matching atoms of both proteins. This score comes from the output of the `OESequenceAlignment` class, where a larger score indicates a better sequence alignment, and scores below a small threshold (around 200) should be considered a bad alignment.

---

**Note:** All structural superposition methods in the `OEStructuralSuperposition` class have a correspond-

ing RMSD value for superposition that can be loosely associated with the quality of the superposition. The `OESecondaryStructureSuperposition` class does not have an RMSD value, but instead uses the Tanimoto score from the underlying shape overlap calculation.

### 1.3.5 How to Correctly Read a PDB File

Reading a PDB file correctly for use in subsequent modeling tasks can be challenging. To correctly read a PDB, one must be aware that PDB header information as well as information about alternate location codes within the PDB file will be lost unless a specific combination of PDB-centric `OEIFlavor`'s are used. Furthermore, the protein itself must be processed by `OEAltLocationFactory` in order to create a molecule with all alternate location atoms retained. With that in mind, we recommend reading PDB files for use in **OESpruce TK** using the following pattern as shown in **ReadProteinFromPDB** below:

**Note:** The mmCIF reader was designed to read all the necessary input by default, and therefore no specific `OEIFlavor`'s are necessary to read them.

CHAPTER

TWO

# SPRUCE

## 2.1 SPRUCE

### 2.1.1 Overview

*SPRUCE* is an application that prepares biomolecular structure files (PDB, mmCIF, oeb, etc.) for downstream modeling applications, such as running a *BROOD* query, docking a ligand with *FRED* or *HYBRID*, predicting poses with *POSIT*, or for running *SZMAP* to analyze active-site hydration thermodynamics. The preparation workflow from an experimental or modeled structure involves, 1) expansion of the asymmetric unit to the biological unit (if structure is from an X-ray crystallography experiment and it it is necessary), 2) enumeration (default) or collapse of alternate locations, 3) building missing pieces e.g. partial sidechains and capping chain breaks, 4) placement and optimization of hydrogen atoms including tautomer enumeration of ligands and cofactors, as well as evaluation of those tautomer states in the biomolecule structure.

### 2.1.2 Spruce Output Naming Conventions

Output files generated from Spruce follows a general naming scheme.

{StructureID}_{ChainIDs}[alt{AltCode}]__DU__{LigandCode}_{ChainID}-{ResidueNumber}.oedu

The scheme can be prefixed by the user using the *-prefix* input parameter.

Example of a structure without an alternate location, 2 files generate due to two ligand sites detected:

`4ICL_AB__DU__T27_A-601.oedu`

`4ICL_AB__DU__14N_A-607.oedu`

Example with an alternate location code:

`1XDN_AaltA__DU__ATP_A-501.oedu`

Apo pocket example:

`1HHP_AB__DU__apo_ASP_A-25.oedu`

Peptidic ligand example, where the ligand section is named slightly differently following the scheme chainID-StartingResidueNumber-1LetterCodes-EndingResidueNumber. Only standard residue get 1 letter codes, whereas non-standard residue will have 3 letter codes separated by dashes.

`4OBD_AB__DU__E-2-PGNFFQNRP-10.oedu`

### 2.1.3 Example Commands

The simplest way to run **SPRUCE** is by passing in an input biomolecular structure file (PDB or mmCIF) as the only argument.

```
prompt> spruce -in 3fly.pdb
```

Here, **SPRUCE** will correctly split and prepare `3fly.pdb` with the *-in* parameter and the standard set of default options. The output will be a set of OEDesignUnit (.oedu) files.

There are also many options that can be invoked from **SPRUCE** on the command line. For instance, several options are available to help **SPRUCE** determine the ligand based on molecular features like the number of residues in a peptidic ligand:

So for example:

```
prompt> spruce -in 4obd.pdb -max_lig_residues 20
```

will correctly split and prepare `4obd.pdb`, based on the *-max_lig_residues* which sets the maximum number of residues that can still be considered a ligand (required for larger, peptidic ligands like those in 4OBD).

Alternatively, one may want to prepare a biomolecular structure file based on the structure of a similar reference system. For instance, one may construct an OEDesignUnit with a set of site residues defined using a reference that includes a ligand:

```
prompt> spruce -ref 4OBD.oedu -in 1hhp.cif
```

Here, an mmCIF file (`1hhp.cif`) is prepared using the site defined in the output OEDesignUnit file prepared using the *-ref* option and an output .oedu file from the example above.

To calculate the Iridium classification as a metric of structure quality, one must provide the map file that corresponds to experiment from which the biomolecular structure was derived:

```
prompt> spruce -map 3rox.mtz -in 3rox.pdb
```

Here a set of OEDesignUnit files will be produced from `3rox.pdb`, where each oedu file will include its own Iridium score taken from a calculation on the density passed in through the *-map* option (`3rox.mtz`). The Iridium score can be extracted from any OEDesignUnit file using the **spruce TK** API, but it will also be reported in the **spruce** application's output.

Several options in **SPRUCE** are available to help control how the system is prepared. For instance by default, **spruce** calculates partial charges and adds any hydrogen atoms missing from the experimental structure. However, if you have a have a downstream application that does not require certain time-consuming computations, like calculating partial charges or adding missing hydrogen atoms, you can tell **SPRUCE** to turn these features off:

```
prompt> spruce -protonate false -charge_radii false -in 3tpp.pdb
```

Here, **SPRUCE** will not add partial charges or hydrogen atoms to the output structures during the preparation process of `3tpp.pdb` by using the boolean options *-protonate* and *-charge_radii*, both of which are set to `true` by default.

To prevent continually over-writing output files, the *-prefix* input parameter. allows you to give unique names to these files.

```
prompt> spruce -in 4icl.pdb -prefix FOO_
```

will write the output structures into a files named `FOO_4ICL_AB__DU__14N_A-607.oedu` and `FOO_4ICL_AB__DU__T27_A-601.oedu`, each identifying separate binding sites.

Finally, to help **spruce** identify ligands in the biomolecular structure file, use the *-ligand_names* flag as in this example with `3b3n.pdb`.

```
prompt> spruce -ligand_names JI2 -in 3b3n.pdb
```

### 2.1.4 Command Line Help

A description of the commandline interface can be obtained by executing **SPRUCE** with the *--help* option.

```
prompt> spruce --help
```

will generate the following output:

```
Help functions:
  spruce --help simple      : Get a list of simple parameters
  spruce --help all         : Get a complete list of parameters
  spruce --help <parameter> : Get detailed help on a parameter
  spruce --help html        : Create an html help file for this program
```

### 2.1.5 Required Parameters

**-in** <filename>
: File containing one or more 3D biomolecules that will be split into known components and fully prepared for downstream modeling.

  File format for *-in* can be one of:

| File type | Extension |
|---|---|
| OEBinary | .oeb .oeb.gz |
| PDB | .pdb .ent .pdb.gz .ent.gz |
| mmCIF | .cif .cif.gz |

  [keyless parameter 1]

### 2.1.6 Optional Parameters

#### Input Options

**-ligand_names** <names>
: If multiple ligands are in the structure but only a specific ones are desired, they can be specified here. Examples are either the three letter code "LIG", or in case of a peptide "VAL-GLU-TYS-PHE-ALA". Multiple different ligands, should be separated by commas, "LIG,GIL,INH".

**-map** <filename>
: Input electron density map from X-ray crystallography, to be used for Iridium score calculation.

**-metadata** <filename>
: Metadata json file containing OEStructureMetadata. Using this can be a way to indicate desired ligands, tautomers to use for ligands, structure sequence etc, as well as structure title for output file names.

**-ref** <filename>
: Reference OEDesignUnit indicating the proper biological unit and relevant binding site.

**-site_residue** <residue identifier>
: Input option to specify a binding site using a single residue specification if apo (or holo). The format is "name:num:insert code:chainid:[fragment number:alternate location]", e.g. "ASP:25: :A:.*: :", indicating Aspartic acid 25 in chain A, also ensuring both the insert code and alternate location is not set, while the fragment number can be anything (the regex .* fuzzy matcher). The fragment number and alternate location can be left

out, in which case they will automatically be replaced by fuzzy matchers. While you could specify ".*:25: :A", which would indicate residue 25 in chain A, irrespective of the residue type, however, we recommend being as explicit as possible.

### Output Options

**-prefix** <prefix>
Prefix used to name output files, default is blank

**-log** <logfile>
The argument for this flag specifies the name of the log file. This overrides any specified prefix. The default will be spruce_output.log, if no prefix is specified.

**-settings** <settingsfile>
The argument for this flag specifies the name of the settings file. This overrides any specified prefix. The default will be spruce_settings.param, if no prefix is specified.

**-verbose** : Triggers copious logging output

**-write_biounits**
Option to write intermediate Bio-DesignUnits, which can be used with *enumsites* to site to potential allosteric sites later.

[default = false]

### Split Parameters

**-altloc** <method>
Parameter determining how alternate locations are handled. The *primary* option, collapses alternate locations, whereas the *enumerate* option attempts to set detected alternate locations, A, B, etc.

[default = enumerate]

**-cofactor_codes** <codes>
Mechanism to define 3 letter codes that should be recognized as co-factors if not done automatically, e.g. "ATP", or "ATP,NAD"

**-excipient_codes**
Mechanism to define 3 letter codes that should be recognized as excipients if not done automatically, e.g. "GOL", or "GOL,DMS"

**-make_packing_residues**
Option to generate packing residues, both for visualization, but also for Iridium classification

[default = true]

**-min_lig_atoms**
Parameter determining the min number of atoms a ligand molecule can have. A reason to lower this number would be for small fragments that need to be classified as ligands

[default = 8]

**-max_lig_atoms**
Parameter determining the max number of atoms a ligand molecule can have. A reasons to increase this number could be for peptidic ligands

[default = 100]

**-max_lig_residues**
Parameter determining the max number of residues a ligand molecule can have. A reasons to increase this number could be for peptidic ligands

[default = 5]

**-max_sys_atoms**
Parameter limiting the max number of atoms in the entire system that spruce will allow for processing. If this limit is reached e.g. due to a large systems or symmetry expansion resulting in a larger than expected system, spruce will stop processing and return false.

[default = 50,000]

**-target**
Parameter telling the system what the "target" is. This is particularly helpful for systems containing both protein and nucleic acids, where the automated system does not correctly identify the nucleic acid as the target of interest. Allowed values are protein or nucleic.

[default = protein]

### Enumerate Sites Parameters

**-add_interactions**
Option to add OEInteractionHints to the design unit(s)

[default = true]

**-add_style**
Option to add visualization style to the design unit(s)

[default = true]

**-collapse_nonsite_alts**
Option to deduplicate structures with different alternate locations if those alternate locations are far from the binding site

[default = true]

**-duplicate_removal**
Option to deduplicate identical structures resulting from symmetry operations

[default = true]

**-enum_cofactors_sites**
Option to generate design units with sites based on components classified as co-factors

[default = false]

**-restrict_to_refsite**
Option to skip generating design units for sites identified, that do not match a provided reference design unit

[default = true]

**-site_size:** `<value>`
Distance from the ligand used to determine the size of the site

[default = 5.0 (angstroms)]

**-superpose**
Option to superpose generated design units, if multiple. If a reference is provided, the first generated design unit will be superposed onto the reference structure, and subsequent structures onto that one.

[default = true]

**-superpose_method :** <method>
The method to use for superposition

| Method | Description |
|---|---|
| global | Global Sequence Alignment to identify CA pairs |
| site | Global Sequence Alignment to identity CA pairs - focusing on the subset in active site |
| ddm | Superposition using the Distance Difference Matrix method (DDM) |
| sse | Superposition using an overlap of Secondary Structure Elements (SSE) |

### Build Parameters

**-build_cterm_caps**
Option to cap broken c-termini in protein chains

[default = true]

**-build_nterm_caps**
Option to cap broken n-termini in protein chains

[default = true]

**-build_sidechains**
Option to build missing or partial protein sidechains

[default = true]

**-delete_clashing_solvent**
Option to allow build steps to remove clashing solvent

[default = true]

**-rot_coverage** <value>
Coverage of rotamer libraries to use, a lower number can be used to speed up side chain re-building skipping lower probability side-chain rotamers.

[default = 100.0]

**-rot_library** <value>
Rotamer library used for building sidechains and loops. Allowed values are 'richardson2016', 'dunbrack', 'richardson'.

[default = richardson2016]

### Prep Parameters

**-charge_radii**
Option to assign partial charge and radii

[default = true]

**-protonate**
Option add and optimize protons in the system

[default = true]

### Protonation Parameters

**-generate_tautomers**
Option to generate and use tautomers in the hydrogen network optimization

[default = true]

**-het_group_nbr_dist** <value>
Distance between heterogens used to determine optimization clusters for protonation

[default = 3.5 (angstroms)]

**-opt_expt_protons**
Option to optimize hydrogens assigned in the experiment.

[default = false]

### Biological Unit Extraction Parameters

**-bu_superpose**
Option to superpose the biological units

[default = false]

**-max_bu_atoms**
Option to limit the size of BUs processed based on number of atoms

[default = 50,000]

**-max_bu_parts**
Option to limit the size of BUs processed based on number of parts

[default = 24]

**-min_align_score**
Option to specify minimum sequence alignment score

[default = 200]

**-pref_author_record**
Option where the author BIOMT record is preferred over the software generated one

[default = true]

CHAPTER

THREE

# SUPERPOSITION

## 3.1 Superposition

### 3.1.1 Overview

*Superposition* is an application that structurally superposes a given biomolecular structure file (oedu, PDB, mmCIF, oeb, etc.) onto a reference structure (oedu, PDB, mmCIF, oeb, etc.).

### 3.1.2 Example Commands

The simplest way to run **Superposition** is by passing in both a reference and a fit input biomolecular structure file:

```
prompt> superposition -ref 4obd.pdb -fit 1hhp.pdb
```

Here, **Superposition** will fit the protein in `1hhp.pdb` against the reference protein in `4obd.pdb`.

The structural superposition method can be set on the command line with the *-method* option:

For example:

```
prompt> superposition -ref 1a29.pdb -fit 1cll.pdb -method ddm
```

will superpose `1cll.pdb` onto the reference file `1a29.pdb` using the `ddm` (*difference-distance matrix*) method.

To prevent continually over-writing output files, the *-prefix* flag allows you to give unique names to these files.

```
prompt> superposition -ref 1a29.pdb -fit 1cll.pdb -prefix FOO_
```

will write the output structures into a file named `FOO_1cll.pdb`.

### 3.1.3 Command Line Help

A description of the commandline interface can be obtained by executing **Superposition** with the *--help* option.

```
prompt> superposition --help
```

will generate the following output:

```
Help functions:
  superposition --help simple      : Get a list of simple parameters
  superposition --help all         : Get a complete list of parameters
  superposition --help <parameter> : Get detailed help on a parameter
  superposition --help html        : Create an html help file for this program
```

### 3.1.4 Required Parameters

**-fit** <filename>
File containing the 3D biomolecules that will be structurally superposed.

File format for *-fit* can be one of:

| File type | Extension |
|---|---|
| OEDesignUnit | .oedu |
| OEBinary | .oeb .oeb.gz |
| PDB | .pdb .ent .pdb.gz .ent.gz |
| mmCIF | .cif .cif.gz |

**-ref** <filename>
File containing the 3D biomolecules that will act as the reference for the structural superposition.

File format for *-ref* can be one of:

| File type | Extension |
|---|---|
| OEDesignUnit | .oedu |
| OEBinary | .oeb .oeb.gz |
| PDB | .pdb .ent .pdb.gz .ent.gz |
| mmCIF | .cif .cif.gz |

### 3.1.5 Optional Parameters

#### Input Options

**-fit_lig** <filename>
A file containing a ligand to use as a constraint during the superposition. Should only be used if the input is not an OEDesignUnit

**-ref_lig** <filename>
A file containing a reference ligand to use as a constraint during the superposition. Should only be used if the input is not an OEDesignUnit

#### Output Options

**-prefix** <prefix>
Prefix used to name output files, default is blank. To avoid overwriting input files, all files are automatically use the prefix *superposed_* if no prefix is specified.

**-log** <logfile>
The argument for this flag specifies the name of the log file. This overrides any specified prefix. The default will be superposition_output.log, if no prefix is specified.

**-settings** <settingsfile>
The argument for this flag specifies the name of the settings file. This overrides any specified prefix. The default will be superposition_settings.param, if no prefix is specified.

**-verbose** : Triggers copious logging output

#### Parameters

**-method** <method>
The method to use for superposition

| Method | Description |
|---|---|
| global | Global Sequence Alignment to identify CA pairs |
| site | Global Sequence Alignment to identity CA pairs - focusing on the subset in active site |
| ddm | Superposition using the Distance Difference Matrix method (DDM) |
| sse | Superposition using an overlap of Secondary Structure Elements (SSE) |

CHAPTER

FOUR

# ENUMSITES

## 4.1 EnumSites

### 4.1.1 Overview

*EnumSites* is an application that enumerates biounit OEDesignUnit (oedu) files, and it creates individual oedu files for every active site discovered.

### 4.1.2 Example Commands

The simplest way to run **EnumSites** is by passing in an unprepped, biounit OEDesignUnit file (oedu) as the input and a fully prepared OEDesignUnit file as a reference.

```
prompt> enumsites -ref 4OBD_AB__DU__E-2-PGNFFQNRP-10.oedu -in 1HHP_AB__DU__biounit.oedu
```

Here, **EnumSites** will use the biounit OEDesignUnit `1HHP_AB__DU__biounit.oedu` with the *-in* parameter, along with a reference OEDesignUnit file `4OBD_AB__DU__E-2-PGNFFQNRP-10.oedu` with the *-ref* option, in order to create OEDesignUnits from `1HHP_AB__DU__biounit.oedu` using the sites defined in the reference structure. The output will be a set of OEDesignUnit (.oedu) files.

Instead of a fully prepared OEDesignUnit file as a reference, one may simply specify the site residues required to define the new site of interest.

So for example:

```
prompt> enumsites -site_residue "ASP:25: :A" -in 1HHP_AB__DU__biounit.oedu
```

will construct OEDesignUnits based on enumeration of sites from the same biounit OEDesignUnit file as before (`1HHP_AB__DU__biounit.oedu`), but here the definition of the site comes from the *-site_residue* option instead of another fully-prepared OEDesignUnit.

To calculate the Iridium classification as a metric of structure quality with `enumsites`, one must provide the map file that corresponds to experiment from which the biomolecular structure was derived:

```
prompt> enumsites -site_residue "MET:124: :A" -map 1a29.mtz -in 1A29_A__DU__biounit.oedu
```

Here a set of OEDesignUnit files will be produced from `1A29_A__DU__biounit.oedu` and the site defined as `MET:124: :A`, but the Iridium classification can be made since the MTZ file (`1a29.mtz`) was passed in through the *-map* option.

To prevent continually over-writing output files, the *-prefix* flag allows you to give unique names to these files.

```
prompt> enumsites -site_residue "ASP:25: :A" -in 1HHP_AB__DU__biounit.oedu -prefix FOO_
```

will write the output structures into a file named `FOO_1HHP_AB__DU__apo_ASP_A-25.oedu`.

### 4.1.3 Command Line Help

A description of the commandline interface can be obtained by executing **EnumSites** with the *--help* option.

```
prompt> enumsites --help
```

will generate the following output:

```
Help functions:
  enumsites --help simple      : Get a list of simple parameters
  enumsites --help all         : Get a complete list of parameters
  enumsites --help <parameter> : Get detailed help on a parameter
  enumsites --help html        : Create an html help file for this program
```

### 4.1.4 Required Parameters

**-in** <filename>
: File containing a biounit OEDesignUnit that whose active site will be enumerated based on the input reference or site residues input.

  File format for *-in* can be one of:

| File type | Extension |
|---|---|
| OEDesignUnit | .oedu |

  [keyless parameter 1]

**-ref** <filename>
: File containing an OEDesignUnit with a site defined that can be used as a reference structure.

  File format for *-ref* can be one of:

| File type | Extension |
|---|---|
| OEDesignUnit | .oedu |

  **NOTE:** Either *-ref* or *-site_residue* must be set, but not both.

**-site_residue** <filename>
: String containing one or more comma-separated residue strings that have the following format:

  "RESIDUE_NAME:RESIDUE_NUMBER:INSERTION_CODE:CHAIN_ID"

  As an example, this is what the string would look like for an asparagine residue, at position 25, with a blank insertion code, on chain A. "ASP:25: :A"

  **NOTE:** Either *-site_residue* or *-ref* must be set, but not both.

### 4.1.5 Optional Parameters

#### Input Options

**-map** <filename>
: Input electron density map from X-ray crystallography, to be used for Iridium score calculation.

**-ref** <filename>
: Reference OEDesignUnit indicating the proper biological unit and relevant binding site.

**-site_residue** <residue identifier>
: Input option to specify a binding site using a single residue specification if apo (or holo). The format is

"name:num:insert code:chainid:[fragment number:alternate location]", e.g. "ASP:25: :A:.*: :", indicating Aspartic acid 25 in chain A, also ensuring both the insert code and alternate location is not set, while the fragment number can be anything (the regex .* fuzzy matcher). The fragment number and alternate location can be left out, in which case they will automatically be replaced by fuzzy matchers. While you could specify ".*:25: :A", which would incidate residue 25 in chain A, irrespecive of the residue type, however, we recommend being as explicit as possible.

### Output Options

**-prefix** <prefix>
Prefix used to name output files, default is blank

**-log** <logfile>
The argument for this flag specifies the name of the log file. This overrides any specified prefix. The default will be enumsites_output.log, if no prefix is specified.

**-settings** <settingsfile>
The argument for this flag specifies the name of the settings file. This overrides any specified prefix. The default will be enumsites_settings.param, if no prefix is specified.

**-verbose** : Triggers copious logging output

### Enumerate Sites Parameters

**-add_interactions**
Option to add OEInteractionHints to the design unit(s)

[default = true]

**-add_style**
Option to add visualization style to the design unit(s)

[default = true]

**-collapse_nonsite_alts**
Option to deduplicate structures with different alternate locations if those alternate locations are far from the binding site

[default = true]

**-duplicate_removal**
Option to deduplicate identical structures resulting from symmetry operations

[default = true]

**-enum_cofactors_sites**
Option to generate design units with sites based on components classified as co-factors

[default = false]

**-restrict_to_refsite**
Option to skip generating design units for sites identified, that do not match a provided reference design unit

[default = true]

**-site_size** <value>
Distance from the ligand used to determine the size of the site

[default = 5.0 (angstroms)]

**-superpose**
Option to superpose generated design units, if multiple. If a reference is provided, the first generated design unit will be superposed onto the reference structure, and subsequent structures onto that one.

[default = true]

**-superpose_method** <method>
The method to use for superposition

| Method | Description |
|---|---|
| global | Global Sequence Alignment to identify CA pairs |
| site | Global Sequence Alignment to identity CA pairs - focusing on the subset in active site |
| ddm | Superposition using the Distance Difference Matrix method (DDM) |
| sse | Superposition using an overlap of Secondary Structure Elements (SSE) |

CHAPTER

FIVE

# GETSTRUCTURE

## 5.1 GetStructure

### 5.1.1 Overview

*GetStructure* is a simple application to download PDB, MMCIF, and MTZ files if available for a given PDB code. The downloads are from the RCSB website.

Input for the application is the 4 character PDB code identifying the structure. All above formats will be downloaded if available.

### 5.1.2 Example Commands

The **getstructure** utility program is run is by passing in a 4-letter PDB code:

```
prompt> getstructure 4OBD
```

Here, **getstructure** will download `4obd.pdb`, `4obd.cif`, and `4obd.mtz`.

CHAPTER

SIX

# RELEASE NOTES

## 6.1 Release History

### 6.1.1 SPRUCE 1.0.0

*Nov 2019*

This version of **SPRUCE** has been built using **OEToolkits 2019.Oct**.

This is the first release of the **SPRUCE** application suite. **SPRUCE** takes experimental biomolecular structure data in either `PDB` or `mmCIF` formats and prepares the structures for use with downstream modeling applications. The **SPRUCE** application suite consists of the following applications and utilities:

- **SPRUCE** prepares biomolecular structure files (`PDB`, `mmCIF`, `OEB`, etc.) for downstream modeling applications such as docking a ligand with **FRED** or **HYBRID**, predicting poses with **POSIT**, or for running **SZMAP** to analyze active-site hydration thermodynamics.
- **Superposition** structurally superposes a given biomolecular structure onto a reference structure.
- **EnumSites** enumerates biounit OEDesignUnit (`OEDU`) files, and creates individual `OEDU` files for every active site discovered.
- **GetStructure** downloads `PDB`, `mmCIF`, and `MTZ` files if available for a given PDB code. The downloads are from the RCSB website.

CHAPTER

SEVEN

# CITATION

## Symbols

## C



## E



## S