

1 Key Concepts

What is Desmond?	explains the basic ideas underlying Desmond: its abstractions of time, space, chemical systems, molecules, atoms, force fields, and motion.
Using Desmond	describes how Desmond fits into a workflow: the processes of creating structure files for input, configuring Desmond, interpreting its output, customizing its behavior, and extending its functionality.

What is Desmond?

molecular dynamics simulation	Desmond is a suite of collaborating applications for carrying out molecular dynamics simulations. Such simulations model the motion of a collection of atoms — a <i>chemical system</i> — over time, according to the laws of classical physics.
chemical system	A collection of atoms representing such real-world entities as a protein molecule in water undergoing a structural change, or a drug molecule interacting with a protein. Desmond models solvents such as water explicitly, as individual water molecules.
thermodynamic environment	<p>The chemical system exists in a <i>thermodynamic environment</i>, which represents the conditions under which the simulation is carried out. This environment mimics the experimental conditions: whether the temperature or pressure is regulated, for example, or whether the system is isolated so that it cannot exchange energy with its environment.</p> <p>The chemical system occupies a three-dimensional volume of space of a specified size, and each atom is represented by a particle at a specific position in that space. Motion is simulated in discrete timesteps like the frames of a film. From one step to the next, a tiny slice of time goes by, and atom positions update accordingly. Atoms move; time advances; atoms move again. Frame by frame, the simulation builds a movie: for example, a micro-second in the life of a protein.</p>

How the atoms move — in which direction? by how much? — is determined by:

- the initial atom positions and velocities,
- the thermodynamic environment, and
- a *molecular mechanics force field*.

**molecular
mechanics
force field**

A set of functions and parameters that describe the potential energy of the interactions between the particles in a chemical system.

In addition to its position, each particle has an associated charge and atomic number, as well as a list of the bonds that it participates in. Using this information, the force field models the forces exerted on each particle by every other particle, thus determining each particle's acceleration.

performance

Simulations such as Desmond's that use the laws of classical physics can only approximate full quantum-mechanical reality. They bow to the limits of computer performance: solving the full set of quantum mechanical equations would take far too long.

Though merely an approximation, integrating Newton's laws of motion for so many pairs of particles still means a great many computations for each step forward. Molecular dynamics simulations therefore face a dilemma:

For accurate results, the simulation timestep must be short enough to capture the vibrational frequency of the atoms you're modeling. Yet the shorter the timestep, the less simulated time you can compute in a practical period of clock time.

To enhance performance as much as possible, Desmond implements a variety of features. Some, such as an algorithm used to minimize interprocessor communication, are built into Desmond and require no action on your part. Others require you to specify their use; for example, you can run Desmond in parallel, using as many processors as your parallel environment can support. Spreading the many computations among many processors can yield a significant increase in speed.

Still other performance features, however, don't make sense for every simulation; therefore, part of configuring a simulation is to set them as you require. In order to make most effective use of Desmond, then, you'll need to learn certain details about the way it works. Where relevant, such performance issues are noted below and throughout the manual.

**free energy
simulations**

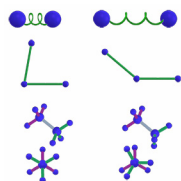
In addition to the simulations described above, Desmond has the ability to perform Gibbs free energy simulations, which compute the change in free energy of a chemical system as it evolves from one state to another. These are described in detail in [Chapter 8 on page 77](#).

Forces

The total force on a particle is the sum of bonded and nonbonded forces.

bonded force A bonded force is a force due to two atoms bound by a shared electron. Bonded forces are of at least three kinds:

Figure 1.1 Bonds



stretch The distance between the centers of two atoms sharing a bond.

bend The angle between two bonds shared by one atom with two other atoms.

torsion The torsion angle between one group of three atoms and another atom, or group of atoms.

In addition, some force fields define other bonded terms.

nonbonded force Nonbonded force is the sum of two forces: *electrostatic* and *van der Waals*. Both kinds of nonbonded forces are a function of the distance between the two atoms.

cutoff radius In principal, electrostatic and van der Waals forces must be computed between every pair of atoms in the system. In practice, however, the magnitude of van der Waals forces falls off rapidly with distance, becoming negligible between pairs of atoms separated by more than a certain distance, referred to as the *cutoff radius*. Therefore, the simulation can restrict van der Waals calculations to only nearby atoms, thus improving performance by reducing the number of computations Desmond must perform.

The cutoff radius cannot be used to limit electrostatic interactions, however, without seriously compromising accuracy. Instead, the electrostatic interactions are split into those between particles within the cutoff radius, and those between more distant particles. Electrostatic interactions are computed explicitly for the closer particle pairs, while the distant particle pairs are computed according to a more efficient method, thus further improving performance.

near interactions Interactions between pairs of particles separated by less than the cutoff radius are called *near interactions*. They comprise both van der Waals forces and the short-range electrostatic forces.

far interactions Electrostatic forces between pairs of particles separated by more than the cutoff radius are referred to as *far interactions*. Instead of computing each pairwise interaction explicitly, Desmond computes far interactions more efficiently in Fourier space, thus:

charge-spreading

1. The application maps charges from particles to nearby grid points needed for the Fourier transform: *charge-spreading*.

2. It computes the interactions in Fourier space.

force interpolation

3. It calculates the resulting forces on the particles from the results at the nearby grid points: *force interpolation*.

Even with optimizations such as the Fourier space computation, far interactions are expensive to compute. They also vary more slowly than the other interactions. You can configure Desmond to compute them less often than near interactions; this is discussed below in [“Dynamics” on page 6](#).

Particles

Desmond represents each atom in the chemical system as a particle. (Special cases for molecules such as water are discussed below; see [“pseudoparticle” on page 4.](#))

- particle The particle:
- models key real-world aspects of an atom: its mass, charge, position, and velocity;
 - participates in specified bonds of specified types; and
 - can be a member of one or more *groups*.

- particle group. You can assign particles to groups for various purposes:
- To understand how energy is distributed throughout the system, particles can belong to different *energy groups*.
 - To control the temperature of subsets of particles independently, particles can belong to different *temperature groups*.
 - To restrain them to a predetermined position relative to another particle group or to the simulated space, particles can belong to a *center-of-mass group*.
 - To hold them motionless in the simulation, particles can belong to *the frozen group*.
 - To define a ligand, used in free energy simulations, particles can belong to *the ligand group*.

Force fields

A force field is a model of the potential energy of a chemical system. It's a set of functions and parameters used to model the potential energy of the system, and thereby to calculate the forces on each particle.

To accurately answer different kinds of questions, Desmond supports several variants of the Amber, CHARMM, and OPLS-AA force field models. See details in [“Available force fields” on page 37.](#)

- pseudoparticle To more accurately simulate the behavior of water or other molecules, certain force fields add electrostatic or van der Waals charges located where no atom is. Desmond implements these as *pseudoparticles*. Desmond supports the most common kinds of pseudoparticles, including those needed for common water models such as the SPC, TIP3P, TIP4P, and TIP5P force fields. See details in [“Virtual sites” on page 53.](#)
- Like particles, pseudoparticles have a mass, charge, position, and velocity; however, their mass is often zero.

Space

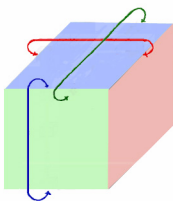
The volume of space in which the simulation takes place is called the global cell.

- global cell A three-dimensional volume of space containing the chemical system. This volume is ordinarily visualized as a three-dimensional rectangle — a *parallelepiped* — though Desmond can simulate other shapes.

The simulation can change dimensions in the course of running — for example, to satisfy a requirement for a constant pressure.

Positions within the global cell are specified in x , y , z coordinates.

Figure 1.2
Periodic boundary conditions



Because complex molecules such as proteins behave differently near a sharp boundary, where they are not fully solvated, Desmond wraps each face of the global cell to its opposite face — a technique known as *periodic boundary conditions*. That is, particles that move leftwards out of the global cell appear to be moving in at a corresponding spot on the right-hand face, and vice-versa; particles that move out the top appear to enter at the bottom, and vice-versa; and finally, particles that move out the front appear at the back, and vice-versa. Thus, you can picture your simulation as an arbitrarily large space tiled by the global cell repeating periodically.

Because the global cell tiles the simulation volume, it must be a shape that can tile a three-dimensional space without gaps, such as a parallelepiped, a hexagonal prism, or a truncated octahedron (a cube with rounded corners).

The global cell also has specified dimensions. It must be large enough that the molecule of interest doesn't interact with its counterparts — its *periodic images* — in other repetitions of the global cell.

When you run a simulation in parallel, Desmond apportions the work among processors by breaking the global cell into smaller boxes. Therefore, how you configure the global cell can have a significant effect on how efficiently your simulation runs in parallel.

Details of these parallelization parameters, and related ones, are discussed in [“Configuration” on page 30](#).

Time

The simulation begins at a specified reference time and advances by timesteps.

reference time The time at which the simulation begins.

Ordinarily, a simulation begins at time 0.0, but it need not. For example, if you wish to use the output of one simulation as the input for the next, thus effectively continuing a simulation, you can specify a reference time equal to the time at which the previous simulation finished.

Starting with the initial chemical system, Desmond:

1. computes forces on each particle based on all the other particles in the system, and
2. moves the particles according to the results of these computations.

This loop is repeated again and again, forming the basis of the timestep.

timestep The period of simulated time computed between each update of the particle positions.

The action of the force field on the atoms is a continuous function which the simulation samples at regular intervals. Thus, the timestep is analogous to the resolution of an image in pixels, or the sampling rate of an analog-to-digital converter. And like those, it presents tradeoffs — too long a timestep sacrifices accuracy; too short, performance.

For accurate results, the timestep must be short enough to capture the vibrational frequency of the atoms you're modeling, typically one femtosecond (fs). Timesteps greater than this cannot model the very fastest vibrations; however, results may be accurate enough for certain purposes. To enable timesteps up to 2.5 fs, Desmond provides *constraints*, discussed in [“Dynamics” on page 6](#).

Dynamics

The action of the force field on the particles is described by a differential equation that Desmond *integrates* — numerically solves — at every timestep, thus computing a new position and velocity for every particle in the system. The differential equation is based on the laws of Newtonian mechanics but different algorithms can vary different parameters, for different purposes. Desmond implements three broad categories:

- Ordinary differential equations that hold certain measures constant — Verlet constant volume and energy, Nose-Hoover constant volume and temperature, MTK constant pressure and temperature, and Piston constant enthalpy.
- Stochastic differential equations that hold certain measures constant and in which one or more of the terms is a stochastic process — Langevin constant volume and temperature, and Langevin constant pressure and temperature. These are sometimes used to correct for possible defects in the underlying data.
- Ordinary differential equations coupled to feedback control systems that keep a certain measure within a certain range — Berendsen constant temperature, and Berendsen constant temperature and pressure.

integrator The particular algorithm that Desmond uses to solve the differential equation is called the *integrator*. Integrators are described in detail in “[Integrator](#)” on page 59.

Desmond allows you to specify other aspects of the motion in your simulation, as well. For example, if you're using certain integrators, you may wish to remove the center-of-mass motion of the chemical system.

timestep scheduling Even with optimizations such as the Fourier space computation, far interactions are expensive to compute. They also change more slowly than the other forces. For many simulations, then, you can improve performance by configuring Desmond to compute the far interactions less often — for example, on alternating timesteps. The integrator still computes the near interactions every timestep, but it skips the far-range computations half the time, weighting the results accordingly to compensate for not including them at every timestep.

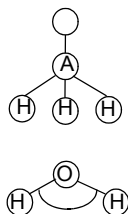
If your simulation can tolerate less accuracy when computing nonbonded near interactions, then those, too, can be scheduled less often. Desmond allows timestep scheduling as follows:

- Bonded forces are computed at every timestep. This is then called the *inner timestep*.
- Nonbonded near forces can be computed at every *n*th timestep, as configured.
- Nonbonded far forces can be computed at the same interval as nonbonded near forces, or a multiple of it. This is then called the *outer timestep*.

Timestep scheduling appears as a configuration parameter called *respa*, an acronym that stands for *reference system propagator algorithm*.

constraints Constraints let you lengthen the timestep by not modeling the very fastest vibrations. Bonds that would vibrate at frequencies faster than a femtosecond are instead held rigid; the integrator moves these constrained particles in unison. A variety of geometries can be constrained this way:

Figure 1.3
Constraints



- a fan of 2–7 particles, each bonded to another particle;
- three mutually connected particles, such as a water molecule.

These constraints are described in detail in [Chapter 6 on page 55](#).

When you prepare your structure file, you specify the types of constraints, if any, and the atoms involved in them. When you configure your simulation, you can specify how precisely to compute the constraints. Whether and how to use constraints depends on simulation-specific factors such as the question you're asking, or the force field you're using.

Using Desmond

Desmond is a suite of collaborating applications. It uses a standard format for input — structure (Maestro) files — and an open format for output — trajectory files, or frame files. So you can also use other applications with Desmond, both public domain and commercial.

Input

Desmond requires two files for input: a structure file that defines the chemical system, and a configuration file that sets simulation parameters.

structure file The *structure file* specifies *what* to simulate: the size of the global cell; the particles it contains, their positions and other properties; the force fields to employ; and possibly more details describing the initial state of the chemical system.

Structure files are also called Maestro files (file suffix `.mae` for Maestro).

configuration file The *configuration file* specifies *how* you want to simulate the chemical system: the reference temperature and pressure, if any; the integrator to use; the length of the timestep; the fineness of the grid to use for charge-spreading; how many processors to assign to a given dimension of the global cell; and possibly many other such parameters.

By using different configuration files with the same structure file, you can run different simulations.

Executables and scripts

Desmond consists of three main executables and two companion Python scripts:

- `mdsim`** The executable that performs the molecular dynamics simulation.
- `minimize`** The executable that prepares the molecular dynamics simulation, if necessary, by minimizing energetic strains in the system so that they don't destabilize the simulation at the first few steps.

You may not need to use **`minimize`** if your system was prepared with care to avoid energetic strains, or if it has already been minimized with another tool.

On the other hand, depending on how the structure file was obtained, you may wish to use **`minimize`** even if you don't intend to run **`mdsim`**, in order to rectify strange conformations resulting from the homology model, or undesired artifacts resulting from x-ray crystallography.

- vr*run** The executable used to analyze framesets output by **mdsim**.
- vip*arr** The Python script that adds force field information to the structure file.
- build_*constraints** The Python script that adds constraint information to the structure file.

Output

Timestep by timestep, an atom traces a path through the global cell as the simulation advances.

- trajectory** The path that molecules take through the global cell is the *trajectory*. Trajectories are written out in a set of files representing a time series, like the frames of a movie.
- frame** Each frame is a file containing the positions and velocities of all the particles and pseudoparticles in the chemical system at that particular timestep. In addition to particle positions and velocities, frames can include system characteristics such as its total energy, temperature, volume, pressure, and dimensions of the global cell.
- You can configure Desmond to output a frame for every timestep, or less often — typically, at an interval corresponding to the outer timestep, when nonbonded far interactions are computed.
- frameset** A time-ordered series of frame files representing the dynamics of the chemical system for the specified time period. Framesets are ordinarily the meaningful unit of analysis for **vr**run or other analysis applications such as **VMD**.

Workflow

The following typical workflow illustrates the roles of Desmond's three main executables, as well as those of other cooperating applications:

- Define the chemical system:
1. Prepare the structure file.
 - a. Typically, start with a Protein Data Base (.pdb) file. Depending on its contents, and the manner in which it was created, it may need some repair of artifacts due to x-ray crystallography. **Maestro** is one tool that can do this; others also exist.
- specify the particles;
- Other starting points are also possible: for example, the results of a different simulation, or a GROMACS file.
- Maestro** or a comparable application outputs a structure file typically containing:
- the *solute*
 proteins, ligands, or other molecules of interest; and
- the *solvent*
 water; and often ions such as sodium, potassium, or chlorine to ensure that the overall chemical system is neutral with respect to charge. (A charge-neutral system is desirable for computing long-range electrostatic interactions.)
- The structure file contains all particle and bond information, but has as yet no information about the force field describing the interactions between particles.
- add the force field;
- b. To add the force field information, the structure file is input to **vip**arr.

- You specify the force field you wish to use, and **viparr** outputs a structure file with the force field information added. It can access a set of databases specifying the required force terms for the various molecules in the chemical system. **viparr** reads the structure file and appends the necessary force terms in a separate section of the file.
- You now have a structure file that describes the particles and forces in your simulation.
- add constraints. c. If you wish to use constraints in your simulation, you now run **build_constraints**. By default, the script constrains the bond length of all bonds involving hydrogen atoms, as well as the angle in all water molecules. The output is a new structure file with the constraint terms added.
- You now have a structure file that describes the particles and forces in your simulation, as well as any constraints you wish to apply.
- Configure the simulation. 4. The simulation still needs to be configured, which involves specifying the values of parameters in a configuration file. The simplest way is to start with an existing configuration file and edit it.
- [Chapter 2 on page 11](#) provides an overview of configuring the simulation. For details about specific configuration file parameters, see the chapters that discuss the applicable configuration file sections.
- Minimize system energy. 5. Most simulations now require that the energy of the system be minimized so that initial forces between atoms are small. (This step is also optional, depending on the initial system characteristics.)
- To minimize the energy of the system, the structure file and associated configuration file are input to **minimize**, which changes the atom positions slightly as needed. It then outputs another structure file but does not change the configuration file.
- Run the simulation. 6. The new structure and the configuration file are now input to **mdsim**, which executes the simulation (possibly for weeks or months), writing the results as frame files at the configured intervals of simulated time.
- Analyze the results. 7. The frameset and configuration file can now be input to **vruntime**, which analyzes the results according to the manner specified in the configuration. For example, you can specify that **vruntime** print the energy of the system for each frame, or the forces on each particle at each frame.
- Other tools such as **VMD**, a freely available visualization application, can be used to analyze results in addition to, or instead of, **vruntime**.

Customizing Desmond

Desmond modularizes its functionality in the form of *plugins*.

plugin A software module that implements a discrete capability, compiled separately so that it can be added to, or removed from, an existing application.

As it runs, a Desmond executable calls plugins as specified in the configuration file. In this way you can execute the code that you need while skipping code that you don't.

Each Desmond executable has a main loop which it repeats: one step in the minimization process, one simulation timestep, or one trajectory frame loaded. Plugins can be called during this loop to perform their work repeatedly as the simulation unfolds. For example, the plugin **eneseq** computes system energy, temperatures, pressures, and other data,

breaking down the energy into various categories, then writes the result to the specified output file.

Or plugins can be called before or after the main loop, to make use of their functionality just once in the simulation, or at longer intervals than once per timestep. For example, **randomize_velocities** reinitializes the velocities of the particles in the simulation according to the Boltzmann distribution for a specified temperature, something you may wish to do once, at the start of the simulation. On the other hand, **trajectory** writes all particle positions to the specified output file at specified intervals, which you probably wish to do more than once, but less often than at every timestep.

Plugins provided with Desmond are described in [“Configuring the built-in plugins” on page 21](#).

extending
Desmond

Desmond already has most or all the functionality required for typical molecular dynamics simulations, but you can extend this functionality by writing your own plugins to, for example, support new force field terms, add new integrators, or apply arbitrary steering forces to the simulation, all without recompiling the Desmond executables.

Implement the functionality you need as a plugin; then specify the parameters for your plugin in the configuration file.

Plugin syntax and other requirements are discussed in [Chapter 9 on page 93](#).