PSim Reference Manual

Release 1.7.0

The PolySwift++ and Composer Teams

May 08, 2018

Contents

Contents:

1 Introduction

1.1 Overview

The *PSim Reference Manual* is a quick-reference manual for PSim users to look up specific features and code block syntax for the compuational engine, PolySwift++. To learn about the complete PSim simulation process, including details regarding input file format and the PSim tutorials, or see examples of using PSim to simulate real-world physics models, please refer to *PSim Quick Start* or *PSim In Depth*.

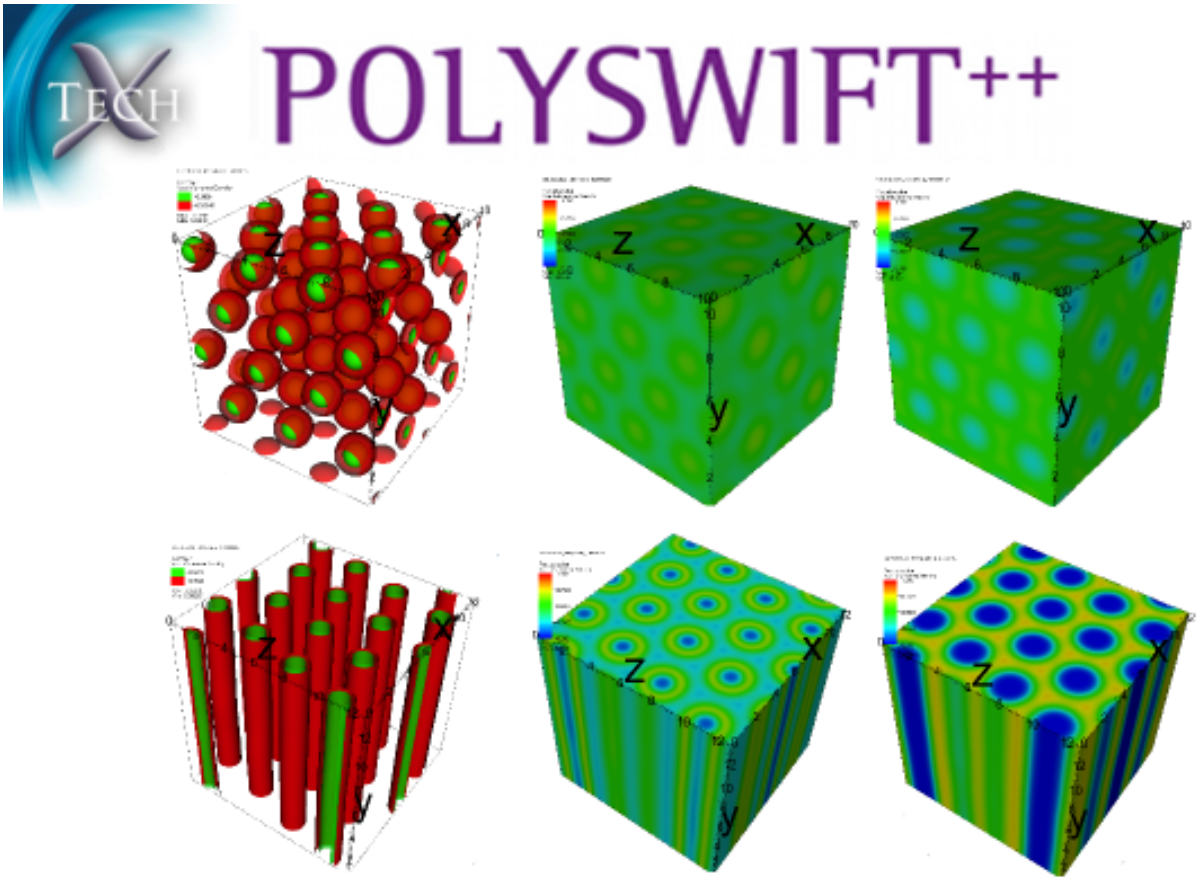
In the following section, we introduce input file basics that are helpful for understanding the descriptions of PSim input file syntax explained in the rest of the *PSim Reference Manual*. In the sections following *Input File Basics*, we describe the various blocks permitted in a PSim input file. The general format of each block's description is:

- Block name
- Summary of the block's purpose
- List of the block's parameters
- Example of the block as used in a PSim input file

We note whenever a block can or should contain another block. For the purpose of this document, a `kind` or a `function` can be considered a parameter of a block if the kind or function can be used to complete the description of the block or modify block characteristics.

Wherever possible, we describe all of the parameters in the same section as the block. While this requires that we repeat some information for different blocks that use the same parameters, you do not need to go elsewhere to find the rest of the block's description. In some cases, where a block's parameters are themselves blocks, we list the parameters' names, however we completely describe the sub-block or block that is nested in its own section.

1.2 Background



PSim provides a flexible simulation tool for studying the mesoscale structure of complex polymeric materials by combining self-consistent field theory (SCFT) methods with high-performance computing. The PolySwift++ engine enables a wide range of systems to be explored, including confined copolymer mixtures and nanocomposites

Self-consistent field theory (SCFT) is a powerful method for studying the complex morphologies of multi-component block copolymers and blend mixtures. The theory enables a systematic coarse-graining of length scales at the molecular level to efficiently describe mesoscale features of phase-separated block copolymer mixtures. Numerical solutions to the SCFT description of these complex fluids, enables these methods to be extended to systems including confinement and nanoparticles.

Code Design

The Polyswift++ engine is an object-oriented code that exploits many of the advanced design principles of the C++ language. The input files are parsed by the TxBase libraries and consist of hierarchical attribute sets. Copolymer architecture, solution methods and boundaries can be specified in a flexible manner from an input file.

SCFT theory

Self-consistent field theory (SCFT) for dense polymer melts[1][2] has been successful in describing complex morphologies for pure block copolymers in bulk. SCFT provides a method whereby the Hamiltonian of a complex system may be transformed into a field theory description, whose mean-field solution is amenable to a battery of analytic and numerical methods. Recently, a numerical method has been used to solve the SCFT equations that does not require foreknowledge about the equilibrium morphologies, allowing more flexibility in applications and greater predictive capability than the fully spectral[3] approach by Matsen. The numerical simulation details are thoroughly discussed in Refs.[4][5][6] and[7].

A field theory transformation allows discrete density operators describing the Hamiltonian of the copolymer system to be replaced by "smeared out" density fields. This transformation permits the complicated chain-chain interactions to be reformulated in terms of a single chain interacting with a chemical potential field ω. The monomer density fields can be composed from restricted chain partition functions $q(\vec{r}, s)$ that may be calculated as the solution to a modified diffusion equation[7] . The $\vec{r}$ argument is a spatial position, and s is a distance along the contour of the polymer chain. Numerical SCFT results are obtained from a fully-flexible chain model which uses a modified diffusion equation to calculate the restricted chain partition functions for each block on the copolymer. In a field theory for fully flexible polymers, the modified diffusion equation for the restricted, single-chain partition function takes on the form:

$$\frac{\partial q(\vec{r}, s)}{\partial s} = R_{g0}^2 \nabla^2 q(\vec{r}, s) - \omega(\vec{r}) q(\vec{r}, s) \tag{1}$$

where R_{g0}^2 is the unperturbed radius of gyration. These diffusion equations are subject to the initial conditions $q(\vec{r}, s) = 1$.

2 Grid

2.1 Grid

`Grid`:

defines the properties of simulation grid.

Grid Parameters

`kind`: specifies the type of grid to use; currently only one option:

`uniCartGrid`: uniform cartesian simulation grid

`cellSizes` (float vector): scaled size of grid cell size in each direction

`numCellsGlobal` (int vector): number of cells in each direction

`decomp` (string): name of a Decomp object used decompose grid in multi-processor simulations

[1] "Functional Integrals and Polymer Statistics", K.F. Freed, Adv. Chem. Phys., 22, p1 (1972).

[2] "Theory of Inhomogeneous Multicomponent Polymer Systems", K.M Hong, J. Noolandi, Macromolecules 14 p727-736 (1981).

[3] "Stable and Unstable Phases of a Diblock Copolymer Melt", M.W. Matsen, M. Schick, Phys. Rev. Lett., 72, p2660-2663, (1994).

[4] "Combinatorial Screening of Complex Block Copolymer Assembly with Self-Consistent Field Theory", F. Drolet, G.H. Fredrickson, Phys. Rev. Lett., 83, 21, p4317-4320, (1999).

[5] "Parallel algorithm for numerical self-consistent field theory simulations of block copolymer structure", S.W. Sides, G.H. Fredrickson, Polymer 44, p5859-5866, (2003).

[6] "Composite mesostructures by nano-confinement", Y. Wu, G. Cheng, K. Katsov, S.W. Sides, J. Wang, J. Tang, G.H. Fredrickson, M. Moskovits, G.D. Stucky, Nature Materials, 3, p816, (2004).

[7] "Field-Theoretic Computer Simulation Methods for Polymers and Complex Fluids", G.H. Fredrickson, V. Ganesan, F. Drolet, Macromolecules, 35, p16-39, (2002).

See also

- *Parallel Domain Decomposition*

3 Domain decomposition

3.1 Parallel Domain Decomposition

`Decomp`:

defines the type of grid decomposition used for multi-processor simulations

Decomp Parameters

`kind`: one of:

`fftw`: uses the decompostion methods for the FFTW slab configuration

`periodicDirs`: The simulation directions to have periodic boundary conditions

See also

- *Grid*

3.2 Decomp Kinds

Slab (FFTW)

`fftw`: kind of **`Decomp`** defines a decompostion for the FFTW slab configuration

fftw Parameters

`transposeFlag` (string , default = 'off'): string flag for selecting the NORMAL or TRANSPOSE layout for the decomp

See also

- *Grid*

4 Surfaces/Nanoparticles

4.1 Boundary

`Boundary`:

blocks define the properties of fields that specify inclusions in an SCFT copolymer simulation. An inclusion is a region of the simulation that excludes polymer/solvent densities by changing the average total density enforced by the incompressibility condition. Blocks derived from the Boundary block can be used to model surfaces, confinement and nanoparticle composite mixtures.

Boundary blocks specify properties of surfaces/particles.

One or more boundary blocks can be created in a simulation.

Boundary Parameters

`kind`: one of:

`fixedWall`: boundary object for a hard stationary surface

`interactingSphere`: boundary object for a spherical nanoparticle that interacts with polymer fields.

`boundaryfield` (string): object name for a PhysField needed for interactions

Boundary Sub-Blocks

In addition to **Boundary** parameters, the **Boundary** block can contain the at least one of the following block(s):

- *Space-Time Function Block*

An 'STFunc' object can be included to specify the spatial dependence of the Flory interaction parameters (kind='flory')

See also

- *Physical Fields*
- *Interaction*

4.2 Boundary Kinds

Fixed Wall/Surface

`fixedwall`: kind of **`Boundary`**. Specifies properties of a stationary substrate or confinement surface

fixedwall Parameters

`STFunc` (block): Space-time function block for specifying the spatial extent of the boundary object

`wallEdgeThreshold` (object): value of scaled incompressibility value for defining the 'edge' of a surface. Used in distance calculations, e.g. between a wall and a nanoparticle.

Example fixedWall Block

```
$ CYLRADIUS = float(NX/2 - int(0.05*NX))
$ CYLCENTER_X = float(NX/2)
$ CYLCENTER_Y = float(NY/2)
$ CYLCENTER_Z = float(1)

# Sets the hyperbolic tangent profile
$ WIDTHPARAM = 1.5

<Boundary softEthyWalls>

  kind = fixedWall
  boundaryfield = totWall

  <STFunc walls>
     kind = pyfunc
     name = cylinder
     paramKeys = [ radius widthParam cylcenterX cylcenterY cylcenterZ ]
     paramValues = [ CYLRADIUS WIDTHPARAM CYLCENTER_X CYLCENTER_Y CYLCENTER_Z ]
  </STFunc>

</Boundary>
```

See also

- *Updater*
- *Physical Fields*
- *Space-Time Function Block*

Spherical Nanoparticles

`interactingSphere`: kind of **`Boundary`**. Specifies properties of a movable spherical nanoparticle.

interactingSphere Parameters

`STFunc` (block): Space-time function block for specifying a region where particles can be dynamically inserted into the simulation domain

`maxNumPtcls` (int): maximum number of particles to be inserted

`updateMovePeriod` (int): number of iterations between particle position update calculation

`updateAddPeriod` (int): number of iterations between particle insertion attempts

`tstepBeforeFirstAdd` (int): number of iterations between particle insertion attempts begin

`ptclNoiseFactor` (float): strength of random spatial displacement [0.0 – 1.0]

`interfaceWidth` (float): controls the size of particle edge (width in a Tanh() profile)

`scaleForceFactor` (float): scales the strength of effective force used in position update algorithm

`radius` (int): radius of a particle in number of simulation cells

`maxdr` (float): maximum displacement allowed for particle update (in R_g)

`boundaryfield` (string): block name of PhysField for specifying interactions with surrounding polymer densities

boundaryFieldThreshold (float): value of incompressiblity field that defines 'edge' of a particle. Used in particle-particle, particle-wall distance calculations

Example interactingSphere Block

This is an example of a Boundary block that specifies parameters for spherical nanoparticles that interact with the polymer chains. These interactions exert effective forces on the nanoparticles, that move the particles. This in turn causes the polymer chains to rearrange. This continues until the system reaches a self-consistent steady state configuration of nanoparticles and polymer fields.

```
$ PTCLRADIUS_SIZE = 0.20 # in units of Rg
$ PTCLRADIUS = int(PTCLRADIUS_SIZE/DX)

$ PTCLWIDTH_SIZE = 0.10 # in units of Rg
$ PTCLWIDTH = PTCLWIDTH_SIZE/DX

$ LI_EDGE = float(0.4 * NX)
$ RI_EDGE = float(NX - LI_EDGE)


<Boundary sphereParticle1>
   kind = interactingSphere
   maxNumPtcls = 3
   updateMovePeriod = 500
   updateAddPeriod = 500
   tstepBeforeFirstAdd = 500
   ptclNoiseFactor = 0.5
   interfaceWidth = PTCLWIDTH
   scaleForceFactor = 10.0
   radius = PTCLRADIUS
   maxdr = 0.25
   boundaryfield = totParticle
   bndryFieldThreshold = 0.0005

   <STFunc insertRegion>
     kind = pyfunc
     name = stripeRegionsX
     paramKeys = [ leftedge rightedge ]
     paramValues = [ LI_EDGE  RI_EDGE ]
   </STFunc>

</Boundary>
```

In this example the Boundary block has a nested STFunc block that controls where nanoparticles are allowed to be inserted into the system as the system relaxes towards a steady-state solution. If this STFunc block 'insertRegion' is removed, nanoparticles are allowed to attempt insertion anywhere within the simulation.

See also

- *Updater*
- *Physical Fields*
- *Space-Time Function Block*

5 Comm

5.1 Comm

`Comm`:

blocks define communications interface for parallel simulations using multiple processors.

Comm Parameters

option:*kind*: one of:

`mpiComm`: Communication interface using the message-passing interface (MPI)

See also

- *Physical Fields*
- *Parallel Domain Decomposition*

6 Interactions

6.1 Interaction

`Interaction`:

blocks define the properties of general interactions for PhysFields in the simulation. One or more of these interactions are held by the EffHamil class container. Returns related quantities for updater objects

Interaction Parameters

`kind`: one of:

`flory`: Flory interaction between two PhysField objects

`floryWall`: Flory interaction between two PhysField object and a wall that interacts with a field with the same form as two bulk density fields

`scfields` (string vector): object names for PhysFields needed for interactions

Interaction Sub-Blocks

In addition to **Interaction** parameters, the **Interaction** block can contain the at least one of the following block(s):

- *Space-Time Function Block*

An 'STFunc' object can be included to specify the spatial dependence of interactions between physical fields. The 'kind=chiCutExpression' can also be used to define an expression parsed directly from the input file. There are other 'STFunc's available, see *Space-Time Function Block*.

```
<STFunc chiramp>
  kind = chiCutExpression
  chi_lower = 0.18
  chi_upper = 0.188
  chi = 0.18 + 0.000025*t
</STFunc>
```

or

```
<STFunc chirzone>
  kind = movTanhSlab
  widthParam = 10.0
  zoneSize = 10.0
  chiNmax = 1.0
  chiNmin = 16.0
  pcomp = 0
  expression = 0.0 + 0.1*t
</STFunc>
```

See also

- *Physical Fields*
- *Effective Hamiltonian*

6.2 Interaction Kinds

Flory Interaction

`flory`: kind of **`Interaction`** specifies the Flory interaction between density fields

flory Parameters

`chi` (float): Flory interaction chi parameter

`STFunc` (object): can hold an object for specifying chi parameters that vary in space and time

Example flory Block

```
<Interaction StyrEthy>
   kind = flory
   chi = 0.12
   scfields = [totStyrDens totEthyDens]
</Interaction>
```

See also

- *Updater*
- *Physical Fields*

- *Space-Time Function Block*

Flory Wall Interactions

`floryWall`: kind of **`Interaction`** specifies the Flory interaction between density field and a wall

floryWall Parameters

`chi` (float): Flory interaction chi parameter

`wallField` (string): Name of field that corresponds to constraint field that is modeling a wall

Example floryWall Block

```
<Interaction EthyWall>
   kind = floryWall
   chi = 0.01
   scfields = [totWall totEthyDens]
   wallField = totWall
</Interaction>
```

See also

- *Updater*
- *Physical Fields*
- *Flory Interaction*

7 General Updaters

7.1 Updater

`Updater`:

> blocks define the properties of general updaters for PhysFields in the simulation. One or more of these updaters are held by the EffHamil class container. The Updater block has access to PhysField and Interaction blocks needed to update a list of PhysField's at each iteration step.

Updater Parameters

`kind`: one of:

> **`steepestDescent`:** Mean-field steepest descent algorithm (with noise) for advancing the chemical potential fields
>
> **`simpleSpecFilter`:** spectral filter algorithm with one global filtering value
>
> **`multiSpecFilter`:** spectral filter algorithm with multiple filtering values
>
> **`poissonUpdater`:** Poisson equation solver

`updateFields` (string vector): object names for PhysFields needed for update

`applyFrequency` (integer): number of update steps between application of updater

`applyStart` (integer): number of update steps before first application of updater

`applyEnd` (integer): number of update steps before turning off updater

See also

- *Physical Fields*
- *Effective Hamiltonian*

7.2 Updater Kinds

Steepest-descent updating of Chemical Potential Fields

`steepestDescent`: kind of **`Updater`** that updates the chemical potential fields for a mean-field polymer SCFT model

steepestDescent Parameters

`relaxlambdas` (float vector): relaxation step sizes r1, r2. Needs r1 > 0 , r2 > 0 and r1 > r2.

`noise` (float): factor for strength of noise term. Typical values should be [0.0 – 0.1]

`updatefields` (string vector): names of PhysFields that steepest descent algorithm acts upon

`interactions` (string vector): names of Interaction's used in Hamiltonian model

Example steepestDescent Block

(bulk only)

```
<Updater wAwB>

   kind = steepestDescent
   type = incompressible
   relaxlambdas = [0.20 0.10]
   noise = 0.002

   updatefields = [totStyrDens totEthyDens]
   interactions = [StyrEthy]

</Updater>
```

See also

- *Updater*
- *Interaction*
- *Physical Fields*

Spectral Filter

simpleSpecFilter: kind of **Updater** uses spectral filter technique to aid relaxing chemical potential fields towards an ordered phase segregated state.

simpleSpecFilter Parameters

cutoffFactor (float): cutoff factor for determining filter strength [0.0 – <1.0]

updatefields (string vector): names of PhysFields that will be filtered

Example simpleSpecFilter Block

```
<Updater specFilter>

  kind = simpleSpecFilter
  applyFrequency = 100
  applyStart = 200
  updatefields = [totStyrDens totEthyDens]
  cutoffFactor = 0.60

</Updater>
```

See also

- *Updater*
- *Physical Fields*

Multiple-value Spectral Filter

multiSpecFilter: kind of **Updater** uses spectral filter technique to aid relaxing chemical potential fields towards an ordered phase segregated state.

multiSpecFilter Parameters

cutoffFactor (float): cutoff factor for determining filter strength [0.0 – <1.0]

updatefields (string vector): names of PhysFields that will be filtered

specCellSizes (int vector): number of cells in [x,y,z] direction defining a filtering region

Example multiSpecFilter Block

```
<Updater specFilter>

  kind = multiSpecFilter
  applyFrequency = 100
  applyStart = 200
  updatefields = [totStyrDens totEthyDens]
```

```
  cutoffFactor = 0.60
  specCellSizes = [4 4 1]

</Updater>
```

See also

- *Updater*
- *Physical Fields*

Poisson-Boltzmann Updater

poissonUpdater: kind of **Updater** uses spectral methods to solve a Poisson equation for a bulk system. Dielectric is assumed homogeneous

poissonUpdater Parameters

bjerrumLen (float): Bjerrum length factor scaled for polyelectrolyte models (see References for details)

updatefields (string): names of charge density PhysFields that will be used in Poisson solve.

Example simpleSpecFilter Block

```
<Updater ePotential>

  kind = poissonUpdater
  bjerrumLen = 0.001
  updatefields = [ totChargeDens ]

</Updater>
```

See also

- *Updater*
- *Physical Fields*
- chargedens

8 EffHamil

8.1 Effective Hamiltonian

EffHamil:

The effective Hamiltonian block is essentially a container for updater and interaction blocks. The combination of interaction and updater blocks defines the model and how the physical fields are manipulated throughout the simulation.

Effective Hamiltonian Parameters

kind: only one:

canonicalMF: mean-field canonical model

updaterSequence: A list of identifiers for Updater objects that lists the order in which they are to be applied

EffHamil Sub-Blocks

In addition to **EffHamil** parameters, the **EffHamil** block must contain at least one of the following blocks:

- *Updater*
- *Interaction*

There is at least one 'Updater' block for the polymer/solvent physical fields. Updater blocks can be added for spectral filtering and/or other auxillary update algorithms.

There is at least one 'Interaction' block for a minimal copolymer model with two monomer species. There should be an 'Interaction' block for each unique combination of two monomer species (e.g. for 3 monomer species in bulk there should be 3 'Interaction' blocks).

Example of EffHamil block

```
<EffHamil mainHamil>

  kind = canonicalMF
  updaterSequence = [wAwB]

  <Updater wAwB>

     kind = steepestDescent
     type = incompressible
     relaxlambdas = [lambda1 lambda2]
     noise = noise_strength
     updatefields = [totStyrDens totEthyDens]
     interactions = [StyrEthy]

  </Updater>

  <Interaction StyrEthy>
     kind = flory
     chi = chiAB
     scfields = [totStyrDens totEthyDens]
  </Interaction>

</EffHamil>
```

9 FFT

9.1 FFT

FFT:

blocks define interface for Fast-Fourier transform implementations.

FFT Parameters

kind: one of:

normalfftw: Implementation of FFT with the FFTW (Fastest Fourier Transform in the West) library set for NORMAL result layout

transposefftw: Implementation of FFT with the FFTW (Fastest Fourier Transform in the West) library set for TRANSPOSE result layout

9.2 FFT Kinds

FFTW (normal decomposition)

normalfftw: kind of **FFT** that defines a normal plan layout for the FFTW package. The plan layout determines how the k-space results are organized on the input simulation grid. The normal type returns results for a slab decomposition along the x-direction for Cartesian grids

normalfftw Parameters

See also

- *Parallel Domain Decomposition*
- *Grid*

FFTW (transpose decomposition)

transposefftw: kind of **FFT** that defines a transpose plan layout for the FFTW package. The plan layout determines how the k-space results are organized on the input simulation grid. The tranpose type returns results for a slab decomposition along the y-direction for Cartesian grids.

transposefftw Parameters

See also

- *Parallel Domain Decomposition*
- *Grid*

10 Physical Fields for SCFT models

10.1 Physical Fields

PhysField:

The physical fields are the primary data-containing blocks in Polyswift++. They contain a field for the main observable quantity and its corresponding conjugate potential within the self-consistent field theory.

Physical Field Parameters

kind: one of:

monomerDens Monomer density field from monomers on polymer chains or those from solvent species

constraint Field that enforces the incompressibility condition

type: Specific parameter for physical fields that set the datatype and dimension of the underlying data field

Example of Physical Field block

```
<PhysField totStyrDens>
   kind = monomerDens
   type = fieldD3R
</PhysField>
```

10.2 PhysField Kinds

Monomer Density Fields

monomerDens: kind of **PhysField** that defines a physical field in the SCFT model. Contains a monomer density and the conjugate chemical potential.

monomerDens Parameters

initOption (string): string flag for setting the initial chemical potential field.

random: uniform random values set betweeen [-0.5, 0.5]

Example monomerDens Block

```
<PhysField totStyrDens>
    kind = monomerDens
    type = fieldD3R
    initOption = "random"
</PhysField>
```

See also

- *Physical Fields*

Constraint Fields

constraint: kind of **PhysField** that enforces the total density in incompressible SCFT models. Contains a pressure field and (for boundary models) a masking field that denotes location of hard surfaces/nanoparticles.

constraint Parameters

`excludeFields` (string vector): names of PhysFields for which constraint applies

`preconditionFactor` (float): value of precondition factor for wall constraints

See also

- *Physical Fields*
- *Monomer Density Fields*

11 Polymer

11.1 Polymer

`Polymer`:

blocks are one of the simulation "entities" in Polyswift++. A simulation "entity" represents real experimental objects to be studied. A polymer is any object that contains sub-blocks representing contiguous monomer chains

Polymer Parameters

`kind`: one of:

`blockCopolymer`: Monodisperse model of arbitrary architecture of copolymer blocks

`volfrac`: The total volume fraction of all monomers from this type of polymer chain. When there are multiple Polymer or Solvent blocks the total volfrac must = 1.

`length`: This is the number of statistically independent segments comprising this polymer. This scales the value of Flory interaction parameters and charge strengths.

Polymer Sub-Blocks

In addition to **Polymer** parameters, the **Polymer** block must contain the at least one of the following block(s):

- *Block*

One 'Block' object is included for each block on the copolymer object. Note: one 'Block' object corresponds to a homopolymer chain.

11.2 Polymer Kinds

Monodisperse Block Copolymer

`blockCopolymer`:

kind of **`Polymer`** is a container for monodisperse copolymer blocks.

blockCopolymer Parameters

Example blockCopolymer Block

```
<Polymer diblock1>

  kind = blockCopolymer
  volfrac = 1.0
  length = 100

  <Block blockA>
    kind = flexPseudoSpec
    scfield = totStyrDens
    ds = 0.05
    lengthfrac = 0.5
    headjoined = [freeEnd]
    tailjoined = [blockB]
  </Block>

  <Block blockB>
    kind = flexPseudoSpec
    scfield = totEthyDens
    ds = 0.05
    lengthfrac = 0.5
    headjoined = [blockA]
    tailjoined = [freeEnd]
  </Block>
</Polymer>
```

See also

- *Polymer*

12 Polymer Chain Blocks

12.1 Block

`Block`:

define the concept of a linear string of chemically similar monomers with a "head" end and a "tail" end. These blocks exist as space curves in the SCF theory. Kinds of block models include flexible, and semi-flexible. Specific solution models include pseudo-spectral. These blocks are nested in Polymer blocks.

Block Parameters

`kind`: one of:

`flexPseudoSpec`: Fully flexible chain model solved with pseudo-spectral algorithm

`chargedFlexPseudoSpec`: Fully flexible chain model solved with pseudo-spectral algorithm and including a contribution from an electric potentail term in polyelectrolyte models

semiflexibleBlock: Semi-flexible block model (not-fully implemented). Templated over different NDIM parameters

scfield (string): object name of a PhysField that defines the type of monomer in this block.

blockfield (string: optional): object name of a PhysField that tracks contribution to overall density for this block only

lengthfrac (float): Length fraction of this block. Values –> [0.0 – 1.0]. All lengthfrac values for all blocks in a Polymer object must sum to 1.0.

headjoined (string vector): List of block name(s) that the head of this block is(are) connected. If this block end is not connected to any other block then the reserved string "freeEnd"

tailjoined (string vector): Same of headjoined but for the other end of this block

ds (float): Contour step size

See also

- *Physical Fields*
- *Effective Hamiltonian*

12.2 Block Kinds

Flexible Block Model

flexPseudoSpec: kind of **Block** that defines a flexible chain model and pseudospectral solution method.

flexPseudoSpec Parameters

bSegRatio (float): ratio of statistical segment length to the segment length in the polymer block that defines the simulation length scale

Example flexPseudoSpec Block

```
<Block blockA>
  kind = flexPseudoSpec
  scfield = totStyrDens
  ds = 0.05
  lengthfrac = 0.5
  headjoined = [freeEnd]
  tailjoined = [blockB]
</Block>
```

See also

- *Physical Fields*
- *Polymer*

13 Solvent

13.1 Solvent

`Solvent`:

> blocks are one of the simulation "entities" in Polyswift++. A simulation "entity" represents real experimental objects to be studied. A solvent is any object that contains a self-consistent field that describes how a monomer-like species affects the mixtures being simulated

Solvent Parameters

`kind`: one of:

> **`simpleSolvent`:** a simple uncharged solvent species that interacts through a Flory-type interaction

`volfrac`: The total volume fraction of this species. When there are multiple Polymer or Solvent objects the total volfrac must = 1.

`scfield`: object name of a PhysField that defines the type of monomer associated with this solvent entity.

13.2 Solvent Kinds

Simple Neutral Solvent

`simpleSolvent`:

> blocks simulate a model of a solvent such that each solvent unit has the same volume as a monomer on the polymer chains. This is a neutral solvent model.

simpleSolvent Parameters

Example simpleSolvent Block

```
<Solvent solvent1>
  kind = simpleSolvent
  volfrac = 0.20
  scfield = totSolDens
</Solvent>
```

14 History Diagnostics

14.1 History

`History`:

> top-level block for recording data from a PSim simulation. You can use History blocks to calculate and record data about a simulation.

Throughout a simulation run, the PolySwift++ engine writes the History data to an HDF5 file (that is separate from the HDF5 files that contain raw field and particle data). The History data file has the file name structure 'runname'_History.h5.

When a simulation is started or restarted, the PolySwift++ engine looks for an existing History file. If a History file is found, new data is appended to the contents of that existing file.

If the PolySwift++ engine does not find a History file, a new file is created. Generally, histories collect data at the end of each time step, after all other blocks have been updated.

History Parameters

kind: one of:

freeEnergy: Excess free-energy in a copolymer mixture

floryConstChi: for a simulation w/spatially constant chi, records chi value due to presence of an STFunc that could be changing the value at each update step

updatePeriodicity (int): number of update steps between recording diagnostic value

tstepBeforeStart (int): number of update steps before start of updating history

tstepBeforeFinish (int): number of update steps before history stops recording

See also

- *Effective Hamiltonian*
- *Interaction*

14.2 History Kinds

Bulk Free-Energy

freeEnergy: kind of **History** that records the excess free-energy for a copolymer mixture

freeEnergy Parameters

updaterName (string): string name for Updater block needed to calculate the free-energy

Example freeEnergy Block

```
<History freeE1>
  kind = freeEnergy
  updatePeriodicity = 10
  updaterName = wAwB
</History>
```

See also

- *History*

Record Flory Chi (spatially homogeneous)

`floryConstChi:` kind of **`History`** that records the changing value of spatially homogeneous Flory chi parameter

floryConstChi Parameters

`interactionName` (string): string name for Interaction block needed to record chi value

Example floryConstChi Block

For a simulation with the following Flory interaction block

```
<Interaction EthyCarb>
  kind = flory
  scfields = [totEthyDens totCarbDens]
  printdebug = DBINTERACTION

  <STFunc chiramp>
    kind = chiCutExpression
    chi_lower = 0.18
    chi_upper = 0.22
    # This will add to the default chi above
    chi = 0.18 + 0.00001*t
  </STFunc>

</Interaction>
```

this History block will record current chi value

```
<History chiN_EC>
  kind = floryConstChi
  updatePeriodicity = 10
  interactionName = EthyCarb
</History>
```

See also

- *History*
- *Interaction*

15 STFunc Block

15.1 Space-Time Function Block

`STFunc`: Space-Time-Function block. Many blocks in PSim can use space-time functions. These are normally contained within other blocks that require time signals or spatial profiles.

When using an STFunc block, place the function in its own STFunc block, specifying the function using the `kind` parameter as in the *Example STFunc Block*.

Most users make use of `kind = expression` to write their own functions.

Descriptions of STFunc parameters are described on each STFunc kind page.

STFunc Kinds

- *General Expression*
- *Expression w/Cutoff*
- *Expression w/Cutoff for Flory Chi*
- *Moving Hyperbolic Tangent Slab*
- *Switchable Moving Hyperbolic Tangent Slab*
- *External Python Function*

Example STFunc Block

In this example the *Interaction* block parameter 'chi' is replaced by the STFunc 'chiramp' which sets the chi values as a function of update step.

```
<Interaction StyrEthy>
   kind = flory
   scfields = [totStyrDens totEthyDens]
   printdebug = DBINTERACTION

   <STFunc chiramp>
      kind = chiCutExpression
      chi_lower = 0.18
      chi_upper = 0.188
      # This will add to the default chi above
      chi = 0.18 + 0.000025*t
   </STFunc>

</Interaction>
```

15.2 STFunc Kinds

General Expression

`expression`: function that can be defined by an arbitrary expression. `expression` is the most versatile STFunc, and uses the parameter `expression` to define the expression to be used.

Note: only certain STFunc blocks may be used in specific top-level blocks. See reference manual for each top-level blocks for details.

---

expression Parameters

`expression` (string): expression to be evaluated.

Example expression Block

```
<STFunc component0>
  kind = expression
  expression = 100.*sin(2.0e9*t)
</STFunc>
```

Expression w/Cutoff

`cutExpression`: function that can be defined by an arbitrary expression, along with upper and lower bounds

cutExpression Parameters

`expression` (string): expression to be evaluated.

`lowerVal` (float): function lower bound

`upperVal` (float): function upper bound

Expression w/Cutoff for Flory Chi

`chiCutExpression`: function that can be defined by an arbitrary expression, along with upper and lower bounds with names that emphasize use for Flory χ parameters

chiCutExpression Parameters

`expression` (string): expression to be evaluated.

`chi_lower` (float): Flory chi lower bound

`chi_upper` (float): Flory chi upper bound

Example chiCutExpression STFunc Block

```
<STFunc chiramp>
   kind = chiCutExpression
   chi_lower = 0.18
   chi_upper = 0.188
   chi = 0.18 + 0.000025*t
</STFunc>
```

Moving Hyperbolic Tangent Slab

`movTanhSlab`: Defines properties for a moving slab (with normal parallel to grid direction) with one value inside the slab and another value outside the slab and a smooth boundary in-between described by a tanh() profile

movTanhSlab Parameters

kind: movTanhSlab

widthParam: width parameter in the hyperbolic tangent profile of the slab 'edge'

zoneSize: width of the zone region

chiNmax: Flory $chiN$ parameter for the region 'inside' the zone

chiNmin: Flory $chiN$ parameter for the region 'outside' the zone

pcomp: Index of the direction parallel to the zone movement (ie perpendicular to the zone face) (x=0, y=1, z=2)

expression: Mathematical expression that determines the location of the center of the zone as a function of update step (eg expression = 0.0 + 0.1*t)

Example movTanhSlab Block

```
<STFunc chirzone>
  kind = movTanhSlab
  widthParam = 10.0
  zoneSize = 10.0
  chiNmax = 1.0
  chiNmin = 16.0
  pcomp = 0
  expression = 0.0 + 0.1*t
</STFunc>
```

Switchable Moving Hyperbolic Tangent Slab

switchMovTanhSlab:

Defines properties for a moving slab (with normal parallel to grid direction) with one value inside the slab and another value outside the slab and a smooth boundary in-between described by a tanh() profile. The slab orientation switches direction (x–>y—z) once trailing edge of slab is outside simulation domain

Switchable Moving Tanh Slab Parameters

kind: movTanhSlab

widthParam: width parameter in the hyperbolic tangent profile of the slab 'edge'

zoneSize: width of the zone region

chiNmax: Flory $chiN$ parameter for the region 'inside' the zone

chiNmin: Flory $chiN$ parameter for the region 'outside' the zone

zoneBuffers: Size of buffer outside of simulation grid that the zone effectively moves through once it leaves the simulation (one for each cartesian component). Typically, these zones should be large enough to allow the zone to completely leave the simulation before entering by another direction

expression: Mathematical expression that determines the location of the center of the zone as a function of update step (e.g. expression = 0.0 + 0.1*t)

maxSweeps: Maximun number of sweeps the zone makes during the simulation. The default is to continue sweeping throughout the simulation.

Example switchMovTanhSlab Block

```
$ zoneVelocity = 0.10
$ sizeZone = 4.0
$ edgeWidth = 4.0
$ sweepsMax = 3
$ xzoneBuffer = float(NX*1.4)
$ yzoneBuffer = float(NY*1.4)

<STFunc chirzoneY>
  kind = switchMovTanhSlab
  widthParam = edgeWidth
  zoneSize = sizeZone
  chiNmax = 12.0
  chiNmin = 18.0
  zoneBuffers = [ xzoneBuffer yzoneBuffer ]
  expression = -60.0 + zoneVelocity*t
  maxSweeps = sweepsMax
</STFunc>
```

External Python Function

`pyfunc`: function defined by a Python script, which is contained in a separate file. The parameter name is used to point to this file.

pyfunc Parameters

`name` (string): name of the function.

`paramKeys` (string vector , optional): names of parameter to set as keys in Python dictionary

`paramValues` (float vector , optional): values of parameters to set as values in Python dictionary corresponding to the key values above

Example pyfunc Block

Python block in a Python file with same name as the input file.

```
#
# Sine modulated wall on "negative Z" side of system
#
def posZFlatWall(x, y, z, t, params={'widthParam':1.0,'walledge':10}):

  global NX
  global NY
  global NZ

  # Convert dictionary to variable assigments
  globals().update(params)
  r = [x,y,z]
  negEdge = [x,y,0]

  negRdis = rdistance(r,negEdge)
  tanhval = math.tanh((negRdis-walledge)/widthParam)
```

```
  wallValue = (tanhval+1)/2.0
  return wallValue
```

Block in PSim referencing the Python function defined in the Python file.

```
# Defines left(bottom) and right(top) edges
# of flat walls... along Y for both 2D and 3D
# domain decomp
$ LOWEDGE = float(14.0)
$ HIGHEDGE = float(LOWEDGE + 50)
$ WALLEDGE_POSZ = float(NZ - LOWEDGE)

# Sets the hyperbolic tangent profile
$ WIDTHPARAM = float(0.1/DX)

<Boundary topNeutralWall>
  kind = fixedWall
  oundaryfield = totWall
  bndryFieldThreshold = 0.005

  <STFunc walls>
    kind = pyfunc
    name = posZFlatWall
    paramKeys = [ widthParam walledge ]
    paramValues = [ WIDTHPARAM  WALLEDGE_POSZ ]
  </STFunc>

</Boundary>
```