

User Manual for the AcCoRD Simulator  
(Actor-based Communication via  
Reaction-Diffusion)

Adam Noel

Manual Release 1 for AcCoRD v1.3  
July 2018



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Feature Summary . . . . .	1
1.2	Motivation . . . . .	2
1.3	Publications . . . . .	4
1.3.1	Primary Reference . . . . .	4
1.3.2	Supporting References . . . . .	5
1.4	History . . . . .	5
1.5	Outline . . . . .	7
<b>2</b>	<b>Download and Installation</b>	<b>9</b>
2.1	Download . . . . .	9
2.2	Recommended Installation Instructions . . . . .	10
2.3	Compiling from Source (Advanced) . . . . .	11
<b>3</b>	<b>How to Use AcCoRD</b>	<b>13</b>
3.1	Summary for Using AcCoRD . . . . .	13
3.2	AcCoRD Configuration . . . . .	14
3.2.1	Configuration File Format . . . . .	14
3.2.2	Visualizing an Environment . . . . .	15
3.3	AcCoRD Configuration Parameters Lists . . . . .	16
3.3.1	Structure of a Configuration File . . . . .	17
3.3.2	Simulation Control . . . . .	17
3.3.3	Chemical Properties . . . . .	19
3.3.4	Environment . . . . .	23
3.4	Running AcCoRD . . . . .	35
3.4.1	Instructions to Run AcCoRD . . . . .	35
3.4.2	Taking Advantage of Random Number Seeds . . . . .	36
3.4.3	Using a Compute Cluster . . . . .	38
3.5	Understanding AcCoRD Output . . . . .	40
3.5.1	Importing to MATLAB . . . . .	40
3.5.2	Reading AcCoRD Output Files . . . . .	47

3.6	AcCoRD Post-Processing . . . . .	50
3.6.1	Plot Maker . . . . .	50
3.6.2	Video Maker . . . . .	54
<b>4</b>	<b>AcCoRD Configuration Examples</b>	<b>61</b>
4.1	Latest Release Examples . . . . .	61
4.2	Specific Release Examples . . . . .	64
4.2.1	Sample Videos in the AcCoRD Journal Paper . . . . .	64

# Chapter 1

## Introduction

This is the User Manual for AcCoRD (Actor-based Communication via Reaction-Diffusion). AcCoRD is a molecular communication simulator and designed as a generic reaction-diffusion solver for flexible system configuration. Actors are placed as sources (i.e., transmitters) or observers (i.e., receivers) of molecules. Environments can be defined with a combination of microscopic and mesoscopic regions. A sample environment is shown in Fig. 1.1.

In this Chapter, we give a summary of AcCoRD’s features and motivation, list the supporting publications, and provide an outline for the rest of this User Manual.

### 1.1 Feature Summary

A summary of AcCoRD’s primary features is as follows:

- It can use a hybrid of microscopic (track each solute molecule individually) and mesoscopic (i.e., subvolume-based) simulation models to define the environment. Every region is either microscopic or mesoscopic.
- “Actors” are either active molecule sources (i.e., transmitters that release molecules) or passive observers (i.e., receivers). They can be placed within a single region or occupy multiple regions.
- Individual regions are cubes, spheres, or rectangles. Spheres must be microscopic but can be infinite in size.
- Regions can be hollow (i.e., surfaces) and/or placed inside of other regions.

- Molecules can move via diffusion or steady uniform flow.
- Framework for chemical reactions can accommodate reactions such as molecule degradation, enzyme kinetics, reversible or irreversible surface binding, ligand-receptor binding, transitions across boundary membranes, and simplified molecular crowding.
- Independent realizations of a simulation can be repeated any number of times (and on different computers) and then aggregated to determine the average behavior and channel statistics.
- Readable setup and output summary files in JSON format.
- Readable warnings and errors at run time about contradictions or missing information in the configuration file.
- Post-processing tools developed in MATLAB (recommend using R2015a or newer) include video generation and plotting receiver observations.

## 1.2 Motivation

AcCoRD provides a simulation platform with the power of a generic reaction-diffusion solver that facilitates communications analysis. The target application is molecular communication, which is the use of molecules as information carriers. Molecular communication is ubiquitous for signalling in nature. Interest in studying molecular communication comes from two directions:

1. We are interested in the design of synthetic communication networks for environments where radio frequency wireless technologies are not appropriate. Molecular communication could be a suitable alternative in such cases. So, we seek a firm understanding of the fundamental limits of molecular communication. We also need insights into practical system design.
2. We are interested in using communications analysis to gain insight into the function of biological mechanisms that rely on molecular communication, such as quorum sensing in bacterial communities. Such insight could improve understanding of diseases and contribute to the development of new prevention or treatment options.

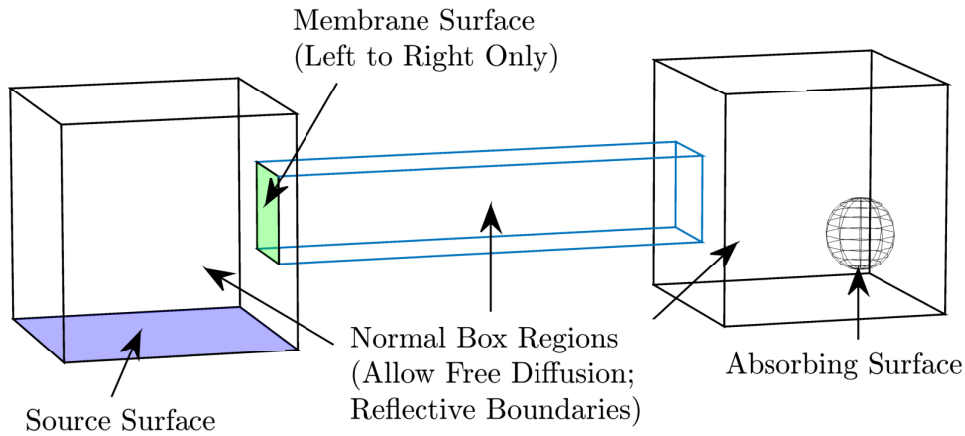


Figure 1.1: Example of a complex simulation environment with many of AcCoRD’s system model components. Reprinted from [1] with permission from Elsevier.

Using diffusion for molecular communication has a number of attractive properties. They include its speed over short distances, its simplicity, and the availability of mathematical models. There are closed-form expressions for the impulse response for diffusion in a number of specific system geometries. However, these systems are usually simplistic and may not accurately represent realistic environments. Having a simulation “sandbox” that is less restricted to particular environments would be a valuable tool.

Existing simulation options (which we discuss in much greater detail in [1]) tend to follow one of two trends:

1. Generic reaction-diffusion solvers are not designed to accommodate the behavior of a transmitter in a communications system, and are not designed to generate the statistics of a communications link.
2. Simulators designed specifically for molecular communication are not built as generic reaction-diffusion solvers. While some are very detailed for their intended environments, they are not flexible for studying new and different environments.

**AcCoRD (Actor-based Communication via Reaction-Diffusion) bridges the gap between reaction-diffusion solvers and molecular communications analysis.** As a reaction-diffusion sandbox, we anticipate that it will contribute the following for molecular communications research:

- **Encourage the use of simulations** without relying entirely on Monte Carlo methods.

- **Increase accessibility** to this multi-disciplinary domain. It can improve the understanding and visualization of known reaction-diffusion environments and their communications channel responses.
- Provide a **platform to verify new analysis** and test transceiver designs.
- **Enable exploration of new environments** that have not or cannot be precisely examined analytically.

## 1.3 Publications

### 1.3.1 Primary Reference

The primary reference paper for the AcCoRD simulator is [1] “Simulating with AcCoRD: Actor-Based Communication via Reaction-Diffusion” (DOI: <http://dx.doi.org/10.1016/j.nancom.2017.02.002>, and is also available from arXiv: <https://arxiv.org/abs/1612.00485>). If you use AcCoRD in your work, then please cite that paper. It provides an overview of AcCoRD, including:

- Motivation for developing a generic reaction-diffusion solver for communication analysis
- Details of all simulation algorithms
- Derivation of computational complexity
- Verification of accuracy by comparing simulation output with analytical expressions
- Insights into appropriate simulation parameters

The paper also includes videos in the supplementary materials. A playlist of these videos is also available here: <https://www.youtube.com/playlist?list=PLZ7uYXG-7XF8UyhFrIuQIiZig1XA89e3i>.

We welcome hearing about how AcCoRD has been used in your work. Where appropriate, we would be happy to add a link to your work on the AcCoRD website (<https://warwick.ac.uk/fac/sci/eng/staff/ajgn/software/accord/>).



### 1.3.2 Supporting References

Papers that describe new features added to AcCoRD:

- A. Noel and D. Makrakis, Algorithm for Mesoscopic Advection-Diffusion, submitted to IEEE Transactions on NanoBioscience in May 2018 (arXiv preprint <https://arxiv.org/abs/1805.12438>). This paper describes the integration of advection (flow) into the reaction-diffusion simulations of the mesoscopic regime. The implementation was added in AcCoRD v1.1.

Portion's of AcCoRD's design was initially motivated in the following papers:

- [2] A. Noel, K. C. Cheung, and R. Schober, On the Statistics of Reaction-Diffusion Simulations for Molecular Communication, in Proc. ACM NANOCOM 2015, Sep. 2015. (DOI: <http://dx.doi.org/10.1145/2800795.2800821>; arXiv preprint <http://arxiv.org/abs/1505.05080>). The simulations in this paper were completed with AcCoRD v0.1, which was an early 2D build.
- [3] A. Noel, K. C. Cheung, and R. Schober, Multi-Scale Stochastic Simulation for Diffusive Molecular Communication, in Proc. IEEE ICC 2015, pp. 2712–2718, Jun. 2015. (DOI: <http://dx.doi.org/10.1109/ICC.2015.7248471>; arXiv preprint <http://arxiv.org/abs/1412.6135>). The simulations in this paper were completed with proof-of-concept MATLAB code that implemented hybrid diffusion with rudimentary transition rules.

## 1.4 History

The plan to build AcCoRD began in 2014, while the developer (Adam Noel) was completing his PhD at UBC. The following timeline is a summary of the major development milestones:

- 2014-09 - developed a proof-of-concept model in MATLAB to describe a hybrid of microscopic and mesoscopic simulation models and how that could be useful for molecular communication simulations. Submitted corresponding paper [3] to the 2015 IEEE International Conference on Communications (ICC) in London, UK.
- 2014-10 - began development of a reaction-diffusion sandbox in C.

- 2015-01 - built a 2D reaction-diffusion solver that would eventually be known as AcCoRD. Presented early results at the Information Theory and Applications workshop in La Jolla, California.
- 2015-04 - added passive and active actors that could be placed "anywhere". Implemented JSON-format (<http://www.json.org/>) simulation output to simplify importing to MATLAB. Most importantly, named simulator AcCoRD (Actor-based Communication via Reaction-Diffusion). Version would become known as v0.1.
- 2015-05 - submitted paper [2] to the 2015 ACM International Conference on Nanoscale Computing and Communication (NanoCom), which demonstrated environments in v0.1.
- 2015-09 - v0.3 upgraded environments to 3D and added the ability to "nest" regions inside of other regions.
- 2016-02 - migrated development to Github for accessibility and issue tracking. v0.4 added spherical regions and actors and improved the tracking of microscopic molecules as they cross region boundaries.
- 2016-05 - v0.5 and its update added surface regions and surface interaction reactions. These reactions include absorption, desorption, and transitions through membranes.
- 2016-05 - v0.6 added bimolecular reactions in the microscopic regime and utilities for visualizing environments and generating video in MATLAB.
- 2016-07 - v0.7 added utilities for plotting passive actor observation signals in MATLAB, and the ability to visualize an environment without simulating it first. Simulations using this version (and its bug-fix update) were used in the primary AcCoRD reference [1].
- 2016-10 - v1.0 added option to set local diffusion coefficients for any region or surface reaction. A number of minor enhancements and fixes brought the major version number from "v0" to "v1".
- 2016-12 - v1.1 added uniform flow, which can be defined globally and also for molecules in specific regions. Corresponding submitted paper is on arXiv (<http://arxiv.org/abs/1805.12438>).
- 2018-05 - v1.2 expanded options for simulation plotting.

- 2018-07 - v1.3 replaced supporting text files with a PDF User's Manual

AcCoRD's change log has a more complete description of the changes made in each version of AcCoRD, starting with v0.2. The change log is included with every download.

## 1.5 Outline

The rest of this User Manual is organized as follows:

- Chapter [2](#) gives instructions on how to download and install AcCoRD. Instructions are also provided for compiling from source.
- Chapter [3](#) describes how to use AcCoRD, including how to prepare a proper configuration file and how to post-process the output.
- Chapter [4](#) describes sample configuration files, including those that are included with the latest version of AcCoRD.



# Chapter 2

## Download and Installation

This Chapter will help you download AcCoRD, install it, and run the sample simulations. AcCoRD has a very simple installation procedure. A complete installation has the following:

1. An AcCoRD executable file that is compiled for your flavor of operating system
2. Directory of MATLAB utilities (optional but recommended for post-processing)
3. One (or more) configuration files to define simulation parameters

The releases hosted on the AcCoRD Download page (<https://warwick.ac.uk/fac/sci/eng/staff/ajgn/software/accord/downloads/>) are provided as prepackaged archives using the recommended file structure in a single zip-file. Each one includes an “optimal” and a “debug” executable, a collection of sample simulations, and a script to run all of the sample simulations. When running AcCoRD, you should use the “optimal” executable. The “debug” executable is useful if you need to run AcCoRD with debugging software, such as gdb (<https://www.gnu.org/software/gdb/>) or valgrind (<http://valgrind.org/>).

### 2.1 Download

Go to the AcCoRD Download page (<https://warwick.ac.uk/fac/sci/eng/staff/ajgn/software/accord/downloads/>) to download the latest stable release or source code. There are also selected previous releases available.

You can also find the latest release, the current code, and older releases on Github (<https://github.com/adamjnoel/AcCoRD/releases>). However, the releases on the Github page keep the executables separate from the source code, so you will need to put the executables in the “bin” folder yourself.

## 2.2 Recommended Installation Instructions

You can install and test a pre-compiled version as follows:

1. **Download AcCoRD** from the AcCoRD Download page (<https://warwick.ac.uk/fac/sci/eng/staff/ajgn/software/accord/downloads/>).
2. **Extract the zip-file** contents to your preferred directory. This completes the installation!
3. **Test your installation:**
  - (a) Open a command line terminal and navigate to the “bin” directory
    - on Windows, run “cmd.exe”
    - on Linux, open a terminal window
  - (b) Run the default simulation:
    - on Windows, enter “accord\_win.exe”
    - on Debian/Ubuntu, enter “./accord\_dub.out”
    - on RHEL/CentOS, enter “./accord\_rc.out”
    - Note: in Linux you may need to give yourself execution permission. If the command does not work, then enter “chmod +x NAME\_OF\_OUT” and try again, where NAME\_OF\_OUT is accord\_dub.out or accord\_rc.out
  - (c) Run all sample simulations (this may take up to 10 minutes):
    - on Windows, enter “run\_accord\_win\_samples.bat”
    - on Debian/Ubuntu, enter “./run\_accord\_dub\_samples.sh”
    - on RHEL/CentOS, enter “./run\_accord\_rc\_samples.sh”
  - (d) By default, simulation output is saved to “bin/results/”. If the directory does not exist, AcCoRD will check up one directory for the existence of a “results” directory, else it will create “bin/results/”.

## 2.3 Compiling from Source (Advanced)

If you want to run AcCoRD on a different operating system, or use different compilation parameters, then you can build it directly from the source code. The source code for the latest release is found on the AcCoRD Download page (<https://warwick.ac.uk/fac/sci/eng/staff/ajgn/software/accord/downloads/>). The source code for the latest and all previous releases is also on the Github AcCoRD Releases page (<https://github.com/adamjgnoel/AcCoRD/releases>). To build AcCoRD, you will need a C-compiler and standard C libraries. The following instructions assume that you have GCC (<https://gcc.gnu.org/>), which is the standard compiler for Linux and can be obtained for Windows via minGW (<http://www.mingw.org/>):

1. **Download AcCoRD's source code** directory (see above)
2. **Extract the zip-file** source code contents to your preferred directory
3. If you want to compile for an OS that is not Windows or Linux, then you should look at modifying source code in two locations. The files "file\_io.c" and "file\_io.h" in the "src" directory each have preprocessor directives that check the existence of "\_linux\_" in order to call the correct function to create a new directory within the current OS. The corresponding code would need to be modified to account for the OS that you want to use.
4. Inspect the build script that you want to use in the "src" directory. The sample build scripts provided correspond to the default installations above (i.e., one "optimal" and one "debug" for each of Windows, Debian/Ubuntu Linux, and RHEL/CentOS Linux). All have filenames that start with "build\_accord". The Windows build scripts are .bat files and the Linux build scripts have no extension. Modify your chosen build script if necessary. Note that the Debian/Ubuntu scripts are actually identical to the RHEL/CentOS build scripts, except for the filenames of the output executable.
5. Open a command line terminal and **navigate to the "src" directory**
6. **Run the desired build script.** The binary will be placed in the "bin" directory (unless you changed the output in the script):
  - sample Windows call: `build_accord_opt_win.bat`
  - sample Linux call: `./build_accord_opt_dub`

- Note: in Linux you may need to give yourself execution permission. If the script does not run, then enter “`chmod +x NAME_OF_SCRIPT`” and try again, where `NAME_OF_SCRIPT` is the name of the script
7. Follow the “Test your installation” steps in the Recommended Installation Instructions above.



# Chapter 3

## How to Use AcCoRD

This Chapter provides basic instructions on how to use AcCoRD. It is up to date for version 1.2. The instructions assume that you have already downloaded and installed AcCoRD. If not, then please refer to Chapter 2. Some of the content in this Chapter is adapted from [1].

### 3.1 Summary for Using AcCoRD

Here is a summary for using AcCoRD on a personal computer. This summary is also a quick start guide. More details for all of these steps can be found throughout the remainder of this Chapter.

1. **Prepare a configuration file** - Many sample files are included with the AcCoRD Download and discussed in Chapter 4. Choose a sample file and modify it as desired. See Section 3.2 for more details, and see Section 3.3 for a detailed listing of all possible simulation parameters.
2. **Run AcCoRD** - Open a command line window and navigate to the AcCoRD “bin” directory. Enter the command to run the simulation. If your OS is Windows and your config file is MY\_CONFIG.txt, then you can enter “accord\_win.exe MY\_CONFIG.txt”. This uses 1 as the default seed number for the random number generator. If your simulation output parameter was set to MY\_OUTPUT, then the files MY\_OUTPUT\_SEED1.txt and MY\_OUTPUT\_SEED1\_summary.txt will be created and placed in the “bin/results” directory. See Section 3.4 for more details.
3. **Import Output in MATLAB** - Set the current directory to the AcCoRD “matlab” directory. If your simulation output parameter was

set to MY\_OUTPUT and written to the “bin/results” directory, then call `accordImport('../bin/results/MY_OUTPUT', 1, true)`. The mat-file MY\_OUTPUT\_out.mat will be created. See Section 3.5 for more details.

4. **Process the Output** - Read the “data” and “config” variables in the file MY\_OUTPUT\_out.mat as desired for post-processing. To plot a figure showing a passive actor signal, modify and run a copy of the file `accordPlotMakerWrapper.m`. To make a video showing the environment and molecules (IF you saved molecule locations in your configuration), modify and run a copy of `accordVideoMakerWrapper`. See Section 3.6 for more details.

## 3.2 AcCoRD Configuration

This Section presents an overview of the AcCoRD configuration file structure and how to preview environments in MATLAB. Once you have prepared a configuration file, you can refer to Section 3.4 to run the simulation.

Every simulation in AcCoRD relies on a single configuration file that defines the simulation environment. Configuration files should be kept in the AcCoRD “config” directory. A number of sample configuration files are included with the AcCoRD download. Modifying an existing configuration is the easiest way to start setting up your own simulation. However, since AcCoRD is a simulation sandbox, the sample files cannot cover all possible configurations. So, this section summarizes the configuration file format. For a detailed description of the structure of a configuration file, including a listing of all possible configuration parameters, please refer to Section 3.3.

### 3.2.1 Configuration File Format

AcCoRD configuration files are written in the JSON (JavaScript Object Notation) interchange format (<http://www.json.org/>). JSON uses a simple structure to store data that is easy to read and modify. You do not need to read all the details of JSON to modify JSON-format files, but here are a few things to keep in mind:

- JSON objects are set of name/value pairs that are enclosed inside curly braces. Each AcCoRD configuration has a single parent object.
- The format for a name/value pair is (including quotation marks) “**Name of pair**”: **VALUE**, where VALUE can be a string (inside

quotes), an integer, a double, another object (inside curly braces), or an array.

- Multiple name/value pairs inside the same object are separated by commas.
- In AcCoRD, placing a comma after the last name/value pair in an object, or writing a decimal number without a leading digit (e.g., .25 for 0.25) will result in an error. Such errors will print out the configuration file from the point where the first error occurred.
- AcCoRD is case sensitive for field names.
- The ordering of fields within a given object does not matter.
- There is no formal commenting mechanism. However, additional string values can be included to behave as comments. This is done throughout the sample configuration files via the “Notes” field (or *any* field name that is not already used for some other purpose).

### 3.2.2 Visualizing an Environment

You can test the environment of a configuration file before simulating it. AcCoRD includes a MATLAB utility for drawing the actors and regions a simulation environment. This utility does **not** do a comprehensive testing of parameter validity, but is useful for a visual confirmation of what the environment will look like. MATLAB version R2015a or newer is recommended.

The “main” function for this utility is `accordEmptyEnvironment.m`, and it is not recommended that you call it directly. The function requires non-trivial input arguments to control how the environment and its contents are drawn. However, a sample wrapper function, `accordEmptyEnvironmentWrapper.m`, is provided that prepares all of the necessary inputs. You should make a copy of the wrapper function and then modify it for your environment. The comments in the wrapper describe all of the arguments that are passed to `accordEmptyEnvironment` and what other files you can refer to for information on modifying the arguments.

There is a sample environment drawing in Fig. 3.1. The configuration file for this environment and the wrapper file to plot it can be found on the AcCoRD Examples page (<https://warwick.ac.uk/fac/sci/eng/staff/ajgn/software/accord/examples/>); refer to the files for the sample videos in the AcCoRD journal paper.

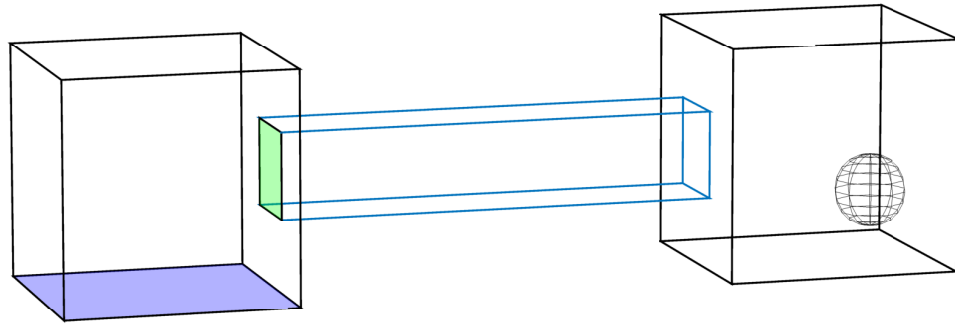


Figure 3.1: Sample empty environment as visualized in MATLAB. Reprinted from [1] with permission from Elsevier.

### 3.3 AcCoRD Configuration Parameters Lists

This Section describes the AcCoRD configuration parameters. Many of the options are labelled as one of the following:

- **usually define** - The parameter has a default value but it should generally be defined by the configuration. A warning will appear if it is not defined (or is defined improperly). It is recommended to fix warning cases because they might prevent importing the simulation output into MATLAB.
- **optional** - The parameter has a default value but it is only needed under certain conditions. A warning will appear if it is not defined and should have been, or if it is defined but not needed. It is recommended to fix warning cases because they might prevent importing the simulation output into MATLAB.

If a parameter has neither label, then it should always be defined. An error will appear if it is not defined. For instructions on using an AcCoRD configuration file to run a simulation, please refer to Section 3.4. For sample complete configuration files, please refer to the AcCoRD Examples webpage (<https://warwick.ac.uk/fac/sci/eng/staff/ajgn/software/accord/examples/>). For a more detailed technical discussion of how these parameters are used within a simulation, please refer to [1]. Flow parameters are described separately in <https://arxiv.org/abs/1805.12438>.

### 3.3.1 Structure of a Configuration File

The overall structure of an AcCoRD configuration file is as follows (with sample values):

```
{
    "Output Filename": "my_output",
    "Warning Override": false,
    "Simulation Control": {},
    "Chemical Properties": {},
    "Environment": {}
}
```

These name/value pairs are used as follows:

- “Output Filename” (usually define) - Prefix for the output data files that will be created by the simulation. Seed information will be appended to this prefix. Default value is “test”.
- “Warning Override” (usually define) - Switch to ignore configuration warnings (when true) and proceed with simulation without pausing. Default value is false. When false, the simulation will pause if any parameters are needed and default values are assigned (either because they were missing or defined incorrectly). Whether true or false, all warning messages will be printed to the command line.
- “Simulation Control” - An object containing global simulation parameters. It should not be empty as shown above!
- “Chemical Properties” - An object containing molecule parameters, including chemical reaction specifications. It should not be empty as shown above!
- “Environment” - An object specifying the regions and actors in the simulation environment. It should not be empty as shown above!

In the remainder of this Section, we provide more details for defining the “Simulation Control”, “Chemical Properties”, and “Environment” objects.

### 3.3.2 Simulation Control

The “**Simulation Control**” object is structured with (some or all of) the following fields (with sample values):

```

"Simulation Control": {
    "Number of Repeats": 1,
    "Final Simulation Time": 0.1,
    "Global Microscopic Time Step": 1e-4,
    "Random Number Seed": 1,
    "Max Number of Progress Updates": 100
    "Small Subvolumes at Hybrid Interface?": true,
    "Max Intrastep Micro to Meso Distance": 10e-6
}

```

These name/value pairs are used as follows:

- **“Number of Repeats” (usually define)** - The number of times that the simulation will be repeated (i.e., the number of realizations) in one execution. Default value is 1.
- **“Final Simulation Time” (usually define)** - The simulated time (in seconds) when the simulation will stop. Not really optional, since the default value is 0. This is not the same as the simulation *run time*, which is the time it takes for a computer to simulate the system.
- **“Global Microscopic Time Step” (usually define)** - The time step for the microscopic regime (in seconds), which is the granularity of a microscopic simulation. Not really optional if there are any microscopic regions, since the default value is 0.
- **“Random Number Seed” (optional)** - Integer used to initialize the random number generator. If you want to aggregate simulation results into a single output, then the different simulations will only be different if they were run with different seeds. However, this parameter is optional because the seed value can be entered via the command line. Default value is 1.
- **“Max Number of Progress Updates” (usually define)** - Integer defining the maximum number of updates on the simulation progress. Updates print the index of the current repeat/realizations being simulated and an estimate of the remaining simulation time. There is no more than one update per realization. Default value is 10.
- **“Small Subvolumes at Hybrid Interface?” (optional)** - Switch to define the transition rules used at the interface between microscopic and mesoscopic regimes (i.e., the hybrid interface). The rules can either assume that mesoscopic subvolumes at the interface are small (and

therefore on the scale of movement in the microscopic regime), or that they are relatively large. This switch is only needed if there is at least one microscopic region and one mesoscopic region. Default value is false. See [1] for more about the ramifications of this parameter.

- **“Max Intrastep Micro to Meso Distance” (optional)** - Maximum distance (in meters) that a microscopic molecule must be within, both before and after a diffusion step, to assume that it could have entered and exited the mesoscopic regime within the time step. This distance is only needed if there is at least one microscopic region and one mesoscopic region. Default value is 0. See [1] for more about the ramifications of this parameter.

### 3.3.3 Chemical Properties

The **“Chemical Properties”** object in an AcCoRD configuration file is structured as follows (with sample values):

```
"Chemical Properties": {
    "Number of Molecule Types": 1,
    "Diffusion Coefficients": [1e-9],
    "Global Flow Type": "Uniform",
    "Global Flow Vector": [1e-3, 0, 0],
    "Does Molecule Type Flow?": [true],
    "Chemical Reaction Specification": []
}
```

These name/value pairs are used as follows:

- **“Number of Molecule Types” (usually define)** - The number of types of molecules that exist in the environment. Default value is 1.
- **“Diffusion Coefficients” (usually define)** - Array of global diffusion coefficients (in meters squared per second) for each type of molecule, each separated by a comma. Length of the array should be equal to the “Number of Molecule Types”. Default value for each coefficient is 0.
- **“Global Flow Type” (usually define)** - String to indicate the default type of flow applied to every region. Possible values are “None” and “Uniform”. If “Uniform”, then flow is steady and uniform. Default type is “None”.

- **“Global Flow Vector” (optional)** - Array defining the global flow parameters. Only needed if “Global Flow Type” is not “None”. If “Global Flow Type” is “Uniform”, then array should have length three and define the flow along the (x,y,z) directions in meters per second.
- **“Does Molecule Type Flow?” (optional)** - Array of switches indicating whether each type of molecule follows the global flow. Only needed if “Global Flow Type” is not “None” and some molecule type does not follow the global flow. Length of the array should be equal to the “Number of Molecule Types”. Default value for each molecule type is true.
- **“Chemical Reaction Specification” (usually define)** - Array containing details of all chemical reactions in the environment. Each reaction has its own object within the array. Default value is an empty array.

If the **“Chemical Reaction Specification”** array is not empty, then its objects are structured with (some of) the following fields (with sample values):

```
{
    "Label": "",
    "Is Reaction Reversible?": false,
    "Reverse Reaction Label": "Some other Reaction Label",
    "Surface Reaction?": false,
    "Surface Reaction Type": "Absorption",
    "Surface Reaction Diffusion Coefficient": 1e-9,
    "Surface Transition Probability": "Steady State",
    "Default Everywhere?": true,
    "Exception Regions": [],
    "Reactants": [0],
    "Products": [0],
    "Products Released?": [false],
    "Release Placement Type": "Leave",
    "Reaction Rate": 1,
    "Binding Radius": 1e-6,
    "Unbinding Radius": 5e-6
}
```

These name/value pairs are used as follows:



- **“Label” (usually define)** - A string ID for the reaction. Only needed if the reaction is reversible in order to indicate which reaction it is coupled to. Default value is an empty string.
- **“Is Reaction Reversible?” (usually define)** - Switch to indicate whether the reaction is reversible. Currently only used when calculating the steady state reaction probabilities of surface reactions. Default value is false.
- **“Reverse Reaction Label” (optional)** - String ID “Label” of the reverse reaction. Only needed if the reaction if “Is Reaction Reversible?” is true. If not defined, then “Is Reaction Reversible?” is set to false.
- **“Surface Reaction?” (optional)** - switch to indicate whether the reaction is a first order surface interaction reaction, such as absorption, desorption, or transitioning through the surface as if it were a membrane. Default value is false.
- **“Surface Reaction Type” (optional)** - String to indicate the type of surface interaction reaction. Possible values are “Normal”, “Absorbing”, “Desorbing”, “Membrane Inner”, and “Membrane Outer”. Only needed if “Surface Reaction?” is true. Default type is “Normal”. The membrane reactions should only be assigned to surface regions that are configured as membranes. The “Membrane Inner” reaction is for molecules that transition from the side of the membrane where the surface is considered to be on the “in” side of the adjacent normal region, i.e., 2D surfaces that are to the left, down, or in directions relative to the normal region (i.e., surface is along the lower x, y, or z face of the normal region), or if the surface is the parent of the normal region. The “Membrane Outer” reaction is for molecules that transition from the side of the membrane where the surface is considered to be on the “out” side of the adjacent normal region, i.e., 2D surfaces that are to the right, up, or out directions relative to the normal region (i.e., surface is along the upper x, y, or z face of the normal region), or if the normal region is the parent of the surface.
- **“Surface Reaction Diffusion Coefficient” (optional)** - Diffusion coefficient (in meters squared per second) to override the default diffusion coefficient for a surface reaction. Only relevant if “Surface Reaction?” is true. The default coefficient is the global diffusion coefficient of the reactant for an absorbing or membrane transition reaction, and that of the first product for a desorption reaction.

- **“Surface Transition Probability” (optional)** - String indicating how the surface reaction probability is calculated. Only needed if “Surface Reaction?” is true and “Surface Reaction Type” is not normal. Options are “Normal”, “Steady State”, and “Mixed”. Default value is “Normal”. See [1] for more about the ramifications of this parameter.
- **“Default Everywhere?” (usually define)** - Switch to indicate whether the reaction exists in all corresponding regions (i.e., in all surface regions for a surface reaction, or all normal regions for a non-surface reaction). Default value is true.
- **“Exception Regions” (usually define)** - Array of strings to indicate the labels of regions that are exceptions to the “Default Everywhere?” value. Default is an empty array.
- **“Reactants” (usually define)** - Array of non-negative integers indicating which molecules are reactants in the reaction. Length of the array should be equal to the “Number of Molecule Types”. Default value for each molecule type is 0. There must be no more than two reactants for most reactions, and no more than one reactant if the reaction is surface reaction that is not “Normal”. The order of the reaction (i.e., zeroth order, first order, or second order) is determined by the sum of the values in this array.
- **“Products” (usually define)** - Array of non-negative integers indicating which molecules are products in the reaction. Length of the array should be equal to the “Number of Molecule Types”. Default value for each molecule type is 0. All values are ignored for “Membrane Inner” and “Membrane Outer” reactions, where it is assumed that the product is the same type of molecule as the reactant.
- **“Products Released?” (optional)** - Array of switches indicating whether each product molecule is detached from the surface into the adjacent normal region. Only needed if “Surface Reaction?” is true. Length of the array should be equal to the “Number of Molecule Types”. Default value for each molecule type is false.
- **“Release Placement Type” (optional)** - String indicating how released molecules are placed. Only needed if “Surface Reaction?” is true and at least one value in “Products Released?” is true. Length of the array should be equal to the “Number of Molecule Types”. Options are “Leave”, “Full Diffusion”, and “Steady State Diffusion”. Default value is “Leave”. See [1] for more about the ramifications of this parameter.

- **“Reaction Rate” (usually define)** - Non-negative number defining the chemical reaction rate. Units are those that are standard for each order of reaction (molecules per second per meter cubed for zeroth order, per second for first order, and meter cubed per molecule per second for second order). Default value is 0.
- **“Binding Radius” (optional)** - Non-negative number defining the maximum distance (in meters) that two reactant molecules in a second order chemical reaction can be separated by at the end of a microscopic time step and still react. Only needed if the reaction is second order and can occur in a microscopic region. Default value is 0.
- **“Unbinding Radius” (optional)** - Non-negative number defining the separation distance (in meters) that is applied to the products of a second order reaction. Only needed if the reaction is second order, it can occur in a microscopic region, and there are at least two products. Default value is If there are exactly two products, then they are placed along the line that joins the locations of the two reactants after they diffused. Each reaction’s displacement from the reaction location is proportional to its diffusion coefficient. If there are more than two products, then each is placed in a random direction but the sum of the distances of the products from the reaction location is equal to the “Unbinding Radius”.

For more information on how these parameters are used to implement chemical reactions, please refer to[1].

### 3.3.4 Environment

The **“Environment”** object in an AcCoRD configuration file is structured as follows (with sample values):

```
"Environment": {
    "Subvolume Base Size": 1e-6,
    "Region Specification": [],
    "Actor Specification": []
}
```

These name/value pairs are used as follows:

- **“Subvolume Base Size”** - Positive number defining the granularity (in meters) of the size of rectangular region subvolumes. Every rectangular region (whether microscopic or mesoscopic) is composed of

square/cubic subvolumes whose length is an integer multiple of “Subvolume Base Size”. This parameter is also used to define the margin of error for detecting regions that overlap incorrectly, so it is needed for any simulation. Default value is 1.

- **“Region Specification”**- Array containing details of all regions in the environment. Each region has its own object within the array. A valid configuration must have at least one region. For details, see below.
- **“Actor Specification”** - Array containing details of all actors in the environment, including active actors and passive actors. Each actor has its own object within the array. A valid configuration must have at least one actor (it could be active or passive). For details, see below.

See below for more about configuring the Region Specification and Actor Specification objects.

## Regions

The objects in the “Region Specification” array of an AcCoRD configuration file are structured with (some of) the following fields (with sample values):

```
{
  "Label": "",
  "Parent label": "",
  "Local Diffusion Coefficients": [0, 0],
  "Shape": "Rectangular Box",
  "Type": "Normal",
  "Surface Type": "Outer",
  "Anchor Coordinate": [0, 0, 0],
  "Anchor X Coordinate": 0,
  "Anchor Y Coordinate": 0,
  "Anchor Z Coordinate": 0,
  "Integer Subvolume Size": 1,
  "Is Region Microscopic?": true,
  "Number of Subvolumes Per Dimension": [1, 1, 1],
  "Number of Subvolumes Along X": 1,
  "Number of Subvolumes Along Y": 1,
  "Number of Subvolumes Along Z": 1,
  "Radius": 5e-6,
  "Local Flow": [
    {
```

```
        "Is Molecule Type Affected?": [true],
        "Flow Type": "Uniform",
        "Flow Vector": [2e-3, 0, 0]
    }
]
```

These name/value pairs are used as follows:

- **“Label” (usually define)** - A string ID for the region. Only needed if another region is nested inside this region or if an actor’s shape will be defined using this region. Default is an empty string. It is recommended that non-empty Region Labels be at least 2 characters long to avoid a JSON bug that treats arrays of single-character strings as one string (which leads to errors if you try to plot the environment).
- **“Parent label” (usually define)** - A string ID stating the region’s parent region, such that this region is nested inside of it (and referred to as the **child**). Needed if this region is nested inside another region. A child region must be entirely inside its parent. Multiple generations of nesting are possible (i.e., a parent region can also have its own parent region, etc.).
- **“Local Diffusion Coefficients” (optional)** - Array of local diffusion coefficients (in meters squared per second) for each type of molecule, each separated by a comma. Length of the array should be equal to the “Number of Molecule Types”. Default value for each coefficient is 0. If local values are not defined, then the values specified by “Diffusion Coefficients” in the “Chemical Properties” object are used.
- **“Shape” (usually define)** - String defining the shape of the region. Options are “Rectangular Box”, “Sphere”, and “Rectangle”. Default is “Rectangular Box”.
- **“Type” (usually define)** - String defining the type of region. Options are “Normal”, “3D Surface”, and “2D Surface”. Default is “Normal”. A “Normal” region occupies the entirety of its volume (except where there are child regions). A “3D Surface” region is a surface to a 3D “Normal” region and a “2D Surface” is a surface to a 2D “Normal” region. For example, a normal “Rectangular Box” could have a surface “Rectangular Box” nested inside and/or a surface “Rectangle” along some or all of one of its faces. Each of the surfaces in this case should be defined as a “3D Surface”.

- **“Surface Type” (optional)** - String defining the type of surface. Options are “Outer”, “Inner”, and “Membrane”. Default is “Membrane”. The “Outer” and “Inner” surfaces are both one-sided and the type specifies the side. These surfaces block diffusion and can have chemical reactions occur at them. An “Inner” surface is considered to be on the “in” side of the adjacent normal region, i.e., 2D shapes that are to the left, down, or in directions relative to the 3D normal region (i.e., surface is along the lower x, y, or z face of the normal region), or if the surface is the parent of the normal region. An “Outer” surface is considered to be on the “out” side of the adjacent normal region, i.e., 2D shapes that are to the right, up, or out directions relative to the 3D normal region (i.e., surface is along the upper x, y, or z face of the normal region), or if the normal region is the parent of the surface. The “Membrane” surface is double-sided and blocks diffusion unless there is membrane reaction defining the rate at which a molecule can pass through. This parameter is only needed if the region’s “Type” is “3D Surface” or “2D Surface”.
- **“Anchor Coordinate” (optional)** - An array with 3 numbers (in meters) that define the region’s location in global space. For a rectangular region, the anchor is its corner with the smallest (X,Y,Z) coordinate. For a “Sphere”, the anchor is the center. This is the default method for defining the anchor; the alternative is to define the (X,Y,Z) values individually. Default is [0, 0, 0].
- **“Anchor X Coordinate”, “Anchor Y Coordinate”, “Anchor Z Coordinate” (optional)** - A number defining the value (in meters) of the region’s location in the corresponding dimension (see “Anchor Coordinate” above). This is the alternative method for defining the anchor; the default is to define the (X,Y,Z) values in a single array. Default value is 0.
- **“Integer Subvolume Size” (optional)** - Positive integer number defining the length of each individual subvolume. The actual subvolume length will be this value times “Subvolume Base Size”. Only needed for rectangular regions (i.e., “Rectangular Box” and “Rectangle”). Default value is 1.
- **“Is Region Microscopic?” (optional)** - Switch indicating whether a region is microscopic or mesoscopic. Only needed for rectangular regions (i.e., “Rectangular Box” and “Rectangle”), since a “Sphere” is always microscopic. Default value is false.

- **“Number of Subvolumes Per Dimension” (optional)** - An array with three non-negative integers defining the total size of the region by specifying its length (in subvolumes) along each Cartesian dimension. Only needed for rectangular regions (i.e., “Rectangular Box” and “Rectangle”). This is the **default** method for defining the length of the region; the alternative is to define the (X,Y,Z) values individually. Default is [1, 1, 1]. A “Rectangle” region must have a single zero value for one of the dimensions.
- **“Number of Subvolumes Along X”, “Number of Subvolumes Along Y”, “Number of Subvolumes Along Z” (optional)** - A non-negative integer defining the length of the region (in subvolumes) along the corresponding dimension. Only needed for rectangular regions (i.e., “Rectangular Box” and “Rectangle”). This is the alternative method for defining the length of the region; the default is to define the (X,Y,Z) values in a single array. Default value is 1.
- **“Radius” (optional)** - A non-negative number defining the radius (in meters) of a “Sphere” region. Only needed for a “Sphere” region. Default value is the “Subvolume Base Size”. Set radius to a value that is bigger than the maximum double (e.g., 1e9999) to have an unbounded region.
- **“Local Flow” (optional)** - An array of objects describing exceptions to the local flow parameters. Array can be of any length but only the first exception specified for a given molecule type will be applied. Each object can have the following:
  - **“Is Molecule Type Affected?” (usually define)** - Array of switches whose length is equal to “Number of Molecule Types”. If switch for any molecule type is true, and the current local flow exception is the first to list that molecule type, then the current exception will apply. Default value for each molecule type is false.
  - **“Flow Type” (usually define)** - String defining type of flow. Same possible values as “Global Flow Type” in the “Chemical Properties” object.
  - **“Flow Vector” (optional)** - Array defining the local exception flow parameters. Only needed if the “Flow Type” for the current exception is not “None”. Same possible values as “Global Flow Vector” in the “Chemical Properties” object.

Here is a sample object for a rectangular region (with irrelevant parameters excluded):

```
{
  "Label": "",
  "Parent label": "",
  "Shape": "Rectangular Box",
  "Type": "Normal",
  "Anchor Coordinate": [0, 0, 0],
  "Integer Subvolume Size": 1,
  "Is Region Microscopic?": true,
  "Number of Subvolumes Per Dimension": [1, 1, 1]
}
```

Here is a sample object for a spherical surface region (with irrelevant parameters excluded):

```
{
  "Label": "",
  "Parent label": "",
  "Shape": "Sphere",
  "Type": "Surface",
  "Surface Type": "Outer",
  "Anchor Coordinate": [0, 0, 0],
  "Radius": 5e-6
}
```

## Actors

The objects in the “**Actor Specification**” array of an AcCoRD configuration file are structured with (some of) the following fields (with sample values):

```
{
  "Is Location Defined by Regions?": false,
  "List of Regions Defining Location": [],
  "Shape": "Rectangular Box",
  "Outer Boundary": [0, 5e-6, 0, 10e-6, 0, 10e-6],
  "Is Actor Active?": true,
  "Start Time": 0,
  "Is There Max Number of Actions?": false,
  "Max Number of Actions": 100,
  "Is Actor Independent?": true,
```



```

    "Action Interval": 5e-3,
    "Is Actor Activity Recorded?": true,
    "Random Number of Molecules?": false,
    "Random Molecule Release Times?": false,
    "Release Interval": 0,
    "Slot Interval": 0,
    "Bits Random?": true,
    "Bit Sequence": [0, 1, 1, 1, 0, 0, 1, 0],
    "Probability of Bit 1": 1,
    "Modulation Scheme": "CSK",
    "Modulation Bits": 1,
    "Modulation Strength": 10,
    "Is Molecule Type Released?": [true],
    "Is Time Recorded with Activity?": false,
    "Is Molecule Type Observed?": [true],
    "Is Molecule Position Observed?": [true]
}

```

These name/value pairs are used as follows:

- **“Is Location Defined by Regions?” (usually define)** - A switch indicating how the shape of the actor is defined. If true, then the actor will exist over the union of a set of regions. If false, then the actor will have its own shape. Default is false.
- **“List of Regions Defining Location” (optional)** - An array of strings of the “Label”s of the regions whose union defines the shape of the actor. The regions can be disjoint. Only needed if “Is Location Defined by Regions?” is true. Default value is an empty array.
- **“Shape” (optional)** - String defining the shape of the region. Options are “Rectangular Box”, “Sphere”, “Rectangle”, and “Point” (however, only active actors can be defined as a point, since molecules in the space of 0 volume cannot be observed). Only needed if “Is Location Defined by Regions?” is false. Default is “Rectangular Box”.
- **“Outer Boundary” (optional)** - A numeric array defining the outer boundary of the actor (in meters). Only needed if “Is Location Defined by Regions?” is false. The length of the array depends on the “Shape” parameter. For “Rectangular Box” or “Rectangle”, the format is [xmin, xmax, ymin, ymax, zmin, zmax]. For “Sphere”, the format is [xCenter, yCenter, zCenter, radius]. The “Sphere” will be unbounded if the

radius is larger than the largest double value, e.g., 1e9999 (but active actors should *not* be unbounded). For “Point”, the format is [x, y, z]. Default value of any element is 0. An actor should be defined within the space occupied by regions. However, a passive actor is permitted to extend beyond the boundary defined by the regions, whereas an active actor is not.

- **“Is Actor Active?” (usually define)** - A switch indicating whether the actor is passive or active. An active actor releases molecules into the environment (i.e., a transmitter). A passive actor observes molecules in the environment (i.e., a receiver). Default is false.
- **“Start Time” (usually define)** - A number indicating the start time (in seconds) of the actor’s behavior. The global simulation start time is 0 and the end time is set by “Final Simulation Time” in the “Simulation Control Object”, so “Start Time” should be between these two values. Default value is 0. Actors that act at the same instant could do so in a random order. In order to impose a specific order, a small offset to an actor’s start time can be used. This can also be done in consideration of the microscopic time step.
- **“Is There Max Number of Actions?” (usually define)** - A switch indicating whether the actor has a preset maximum number of actions. One action is the start of a symbol interval (i.e., modulated data symbol) for an active actor or an observation of molecules by a passive actor. Default value is false.
- **“Max Number of Actions” (optional)** - A positive integer number of actions for the actor to perform. Only needed if “Is There Max Number of Actions?” is true. The actor may perform fewer actions if the “Final Simulation Time” is reached first. Default value is 1.
- **“Is Actor Independent?” (usually define)** - Switch to indicate whether the actor is independent. The actual value is ignored and will be accounted for in a future release. Default value is true, since currently all actors are independent.
- **“Action Interval” (usually define)** - A number indicating the time (in seconds) between actions performed by the actor. Default value is 1.
- **“Is Actor Activity Recorded?” (usually define)** - Switch to indicate whether the actor’s actions are recorded in the simulation output

file. Active actor activity is its sequence of emitted symbols. Passive actor activity is the number of each type of molecule that is observed (and optionally the coordinates of those molecules). Default is true.

- **“Random Number of Molecules?” (optional)** - A switch indicating whether the precise number of molecules released by the transmitter for a given symbol is random or deterministic. Only needed if the actor is active. Default is false. See Table 3.1 for how this parameter controls the release of molecules.
- **“Random Molecule Release Times?” (optional)** - A switch indicating whether the precise molecule release times within a symbol interval are random or deterministic. Only needed if the actor is active, and only used if “Random Number of Molecules?” is true. Default is false. See Table 3.1 for how this parameter controls the release of molecules.
- **“Release Interval” (optional)** - A non-negative number indicating the time (in seconds) that an active actor will release molecules for a given interval. Analogous to pulse width. The value can be larger than the “Action Interval”, i.e., an active actor can be performing multiple actions simultaneously. Only needed if the actor is active. Default value is 0 (i.e., molecules are released instantaneously). See Table 3.1 for how this parameter controls the release of molecules.
- **“Slot Interval” (optional)** - A non-negative number indicating the period (in seconds) between molecule releases within a single “Release Interval”. Only needed if the actor is active, and only used if “Random Number of Molecules?” is false. Default value is 0. See Table 3.1 for how this parameter controls the release of molecules.
- **“Bits Random?” (optional)** - A switch indicating whether the bit sequence, whose symbols are modulated by an active actor, is randomized. Only needed if the actor is active and the modulation scheme is not “Burst”. Default value is true.
- **“Bit Sequence” (optional)** - An array of binary values specifying the bits that are to be modulated by an active actor. Only needed if the actor is active, the modulation scheme is not “Burst”, and “Bits Random?” is false. The length of the sequence should be consistent with “Max Number of Actions” and “Modulation Bits”, otherwise “Max Number of Actions” will be corrected. Default value for each element is 0.

- **“Probability of Bit 1” (optional)** - A value between 0 and 1 that defines the probability that given bit in an active actor’s data sequence will have value 1. Only needed if the actor is active, the modulation scheme is not “Burst”, and “Bits Random?” is true. Every bit is determined independently. Default value is 0.5.
- **“Modulation Scheme” (optional)** - A string defining the modulation scheme used by an active actor. Options are “CSK” (Concentration shift keying) and “Burst”. For “CSK”, the actor will release a number of molecules that is (on average) linearly proportional to the current data symbol. For “Burst”, the actor will release (on average) a constant number of molecules (of possibly multiple types) in every action interval. Only needed if the actor is active. Default value is “CSK”.
- **“Modulation Bits” (optional)** - A positive integer number defining the number of bits in each symbol of an active actor’s data sequence. Only needed if the actor is active and the modulation scheme is not “Burst”. Default value is 1.
- **“Modulation Strength” (optional)** - A positive number defining the strength of an active actor’s molecule emissions. The precise use of this value (as either a rate or an absolute number of molecules) depends on how the actor’s other transmission parameters are defined. Specifically, for “CSK” modulation, the strength is an (average) number of molecules per slot if “Random Molecule Release Times?” is false, and an expected generation rate (in molecules per second) if both “Random Number of Molecules?” and “Random Molecule Release Times?” is true, where the number or rate are scaled by the value of the current symbol value. The behavior for “Burst” modulation is similar, except that the symbol value is always 1 and multiple types of molecules can be released simultaneously. Only needed if the actor is active. Default value is 1.
- **“Is Molecule Type Released?” (optional)** - An array of switches indicating what type(s) of molecule is (are) released by an active actor. Length of the array should be equal to the “Number of Molecule Types”. If the array does not have the correct length, then the first element is set to true. Otherwise, the default value of each element is false. For “CSK” modulation, all true elements after the first are ignored. Only needed if the actor is active.

Table 3.1: Table Listing How Release Parameters are Interpreted

“Random Number of Molecules”	“Random Molecule Release Times”	How “Modulation Strength” is Applied
false	n/a	# of molecules in <b>each</b> slot within release interval
true	false	<b>Expected</b> # of molecules in <b>each</b> slot with release interval
true	true	<b>Expected</b> release <b>rate</b> (in molecules per second) over entire release interval

- **“Is Time Recorded with Activity?” (optional)** - A switch indicating whether the simulation time of each action of a passive actor should be recorded with the rest of the action information. Only needed if the actor is passive. Default is false.
- **“Is Molecule Type Observed?” (optional)** - An array of switches indicating what type(s) of molecule is (are) observed by a passive actor. Length of the array should be equal to the “Number of Molecule Types”. Only needed if the actor is passive. Default value of each element is true.
- **“Is Molecule Position Observed?” (optional)** - An array of switches indicating whether a passive actor should record the position of every molecule that it observes. Length of the array should be equal to the “Number of Molecule Types”. Only needed if the actor is passive. Default value of each element is false. To generate a meaningful simulation video, at least one passive actor should be recording molecule locations. However, doing so generally increases the output file size considerably.

Here is a sample object for an active actor that is defined by two regions:

```
{
  "Is Actor Location Defined by Regions": true,
  "List of Regions Defining Location": ["A", "B"],
  "Is Actor Active": true,
  "Start Time": 0,
  "Is There Max Number of Actions": true,
  "Max Number of Actions": 20,
  "Is Actor Independent": true,
```

```

    "Action Interval": 100e-3,
    "Is Actor Activity Recorded?": true,
    "Random Number of Molecules?": false,
    "Release Interval": 25e-3,
    "Slot Interval": 5e-3,
    "Bits Random?": true,
    "Probability of Bit 1": 0.5,
    "Modulation Scheme": "CSK",
    "Modulation Bits": 1,
    "Modulation Strength": 200,
    "Is Molecule Type Released?": [true, false]
}

```

Here is a sample object for a passive actor whose location is defined manually:

```

{
    "Is Actor Location Defined by Regions?": false,
    "Shape": "Sphere",
    "Outer Boundary": [0, 0, 10e-6],
    "Is Actor Active?": false,
    "Start Time": 1e-10,
    "Is There Max Number of Actions?": false,
    "Is Actor Independent?": true,
    "Action Interval": 1e-3,
    "Is Actor Activity Recorded?": true,
    "Is Time Recorded with Activity?": false,
    "Is Molecule Type Observed?": [true, false],
    "Is Molecule Position Observed?": [false, false]
}

```

## 3.4 Running AcCoRD

This Section has instructions for running AcCoRD simulations, including discussions of random number seeds and running AcCoRD on a compute cluster. These instructions assume that you have already prepared a configuration file. If not, then please refer to Sections 3.2 and 3.3. Once your simulations have been run, you can refer to Sections 3.5 and 3.6 for reading the files and importing into MATLAB.

### 3.4.1 Instructions to Run AcCoRD

Once you have a configuration file ready, then you can try to run it:

1. Open a command line window (e.g., run `cmd.exe` in Windows or open a terminal in Linux).
2. Navigate to the AcCoRD “bin” directory using the “`cd`” command (e.g., enter “`cd \PATH_TO_ACCORD\AcCoRD-1.0\bin\`” on Windows)
3. Run either the optimal or debug executable. Generally, the optimal executable is recommended. The call syntax for each is the same: “`MY_EXECUTABLE MY_CONFIG SEED_VALUE`”, where the executable `MY_EXECUTABLE` is:
  - `accord_win.exe` or `accord_win_debug.exe` on Windows
  - `./accord_dub.out` or `./accord_dub_debug.out` on Debian or Ubuntu Linux
  - `./accord_rc.out` or `./accord_rc_debug.out` on RHEL or CentOS Linux

`MY_CONFIG` is the name of your configuration file, e.g., `my_config.txt`, which can include a relative or absolute directory. AcCoRD will search for `MY_CONFIG` in the following order: relative to the current directory, relative to a “`bin/config`” directory, and finally relative to a “`config`” directory (sibling to “`bin`”). The `SEED_VALUE` is a positive integer random number seed to initialize the random number generator. This seed is optional and will over-ride the seed value specified by the configuration.

While AcCoRD runs, the information printed to the command line will include the following:

1. Version information.

2. Where the configuration file was found (if at all).
3. Warnings from the configuration file parameters. If there are any warnings, and the “Warning Override” property is false, then execution will pause and you will be prompted to continue or cancel the simulation.
4. A summary of the configuration (e.g., number of regions, subvolumes, actors, etc.)
5. Location and name of the two output files that will be created. A “results” folder will be created inside the “bin” directory if it did not exist and if there is no “results” sibling directory. The files will be named MY\_OUTPUT\_SEEDX.txt and MY\_OUTPUT\_SEEDX.summary.txt, where MY\_OUTPUT is the “Output Filename” defined in the configuration file, and X is the seed value. If the output files already exist, then they will be over-written.
6. Simulation progress with estimated time remaining (if there are multiple realizations being simulated)
7. Simulation run time.

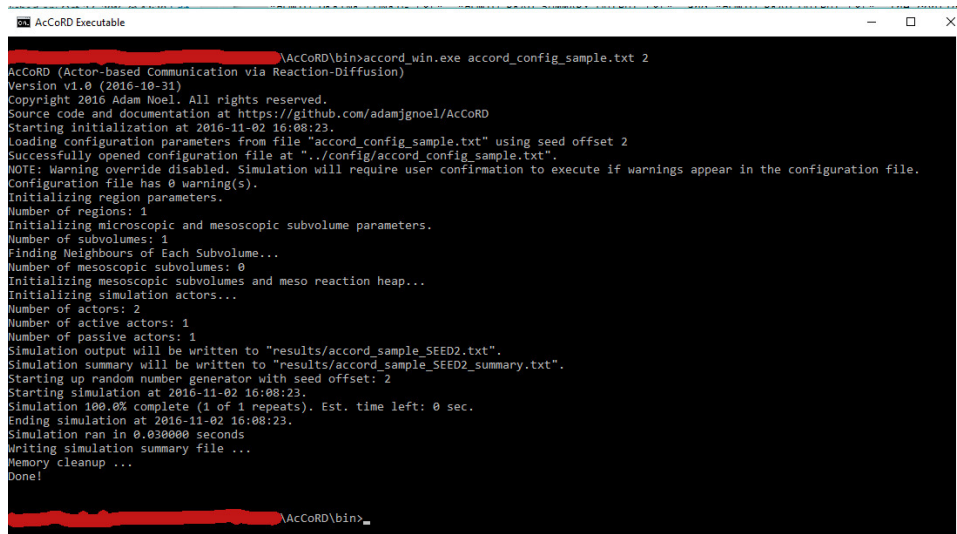
Sample command line outputs are shown without (Fig. 3.2) and with (Fig. 3.3) warnings. If warnings appear, the user is told what action is being taken to address them and the user also has to press “enter” in order to continue the simulation.

### 3.4.2 Taking Advantage of Random Number Seeds

One of the global parameters in an AcCoRD simulation is the random number “seed”. The seed is used to initialize the random number generator (RNG). The RNG drives all of the random behavior in a simulation, such as the values of random bits, the displacement of microscopic molecules, and when chemical reactions occur. A key desired property of an RNG is that it can generate a long sequence of values that are effectively independent, such that someone who knows all the prior values in the sequence cannot guess the next value. Furthermore, sequences that are initialized with different seeds should also be independent. One of the primary features of AcCoRD is that its design facilitates repeating a simulation a large number of times. It can do this in one of two ways:

1. In a single execution, a simulation is repeated multiple times according to the “Number of Repeats” property in the “Simulation Control” object. Each of these realizations should be independent.



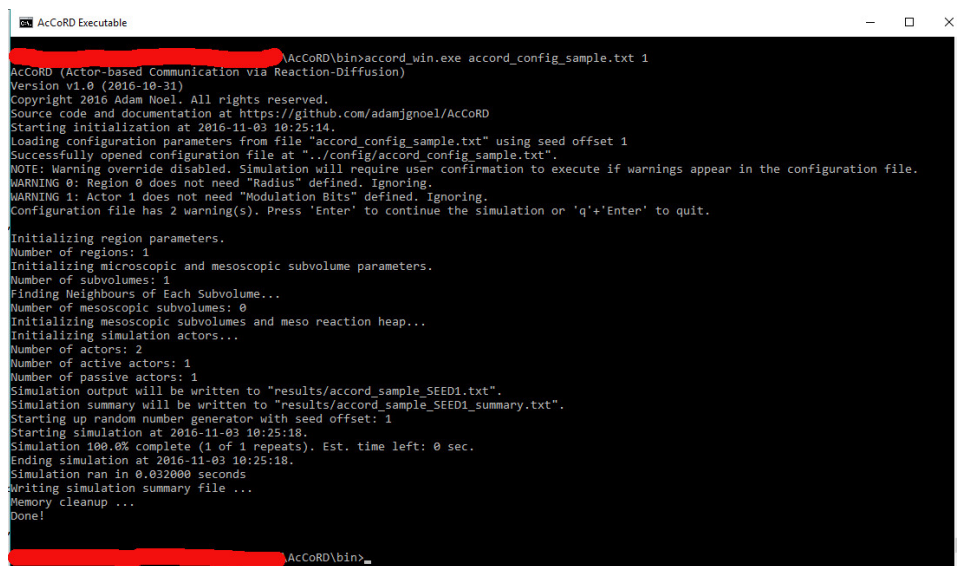


```

AcCoRD (Actor-based Communication via Reaction-Diffusion)
Version v1.0 (2016-10-31)
Copyright 2016 Adam Noel. All rights reserved.
Source code and documentation at https://github.com/adamjnoel/AcCoRD
Starting initialization at 2016-11-02 16:08:23.
Loading configuration parameters from file "accord_config_sample.txt" using seed offset 2
Successfully opened configuration file at "../config/accord_config_sample.txt".
NOTE: Warning override disabled. Simulation will require user confirmation to execute if warnings appear in the configuration file.
Configuration file has 0 warning(s).
Initializing region parameters.
Number of regions: 1
Initializing microscopic and mesoscopic subvolume parameters.
Number of subvolumes: 1
Finding Neighbours of Each Subvolume...
Number of mesoscopic subvolumes: 0
Initializing mesoscopic subvolumes and meso reaction heap...
Initializing simulation actors...
Number of actors: 2
Number of active actors: 1
Number of passive actors: 1
Simulation output will be written to "results/accord_sample_SEED2.txt".
Simulation summary will be written to "results/accord_sample_SEED2_summary.txt".
Starting up random number generator with seed offset: 2
Starting simulation at 2016-11-02 16:08:23.
Simulation 100.0% complete (1 of 1 repeats). Est. time left: 0 sec.
Ending simulation at 2016-11-02 16:08:23.
Simulation ran in 0.030000 seconds
Writing simulation summary file ...
Memory cleanup ...
Done!

```

Figure 3.2: Sample command line output on Windows running one of the sample configuration files.



```

AcCoRD (Actor-based Communication via Reaction-Diffusion)
Version v1.0 (2016-10-31)
Copyright 2016 Adam Noel. All rights reserved.
Source code and documentation at https://github.com/adamjnoel/AcCoRD
Starting initialization at 2016-11-03 10:25:14.
Loading configuration parameters from file "accord_config_sample.txt" using seed offset 1
Successfully opened configuration file at "../config/accord_config_sample.txt".
NOTE: Warning override disabled. Simulation will require user confirmation to execute if warnings appear in the configuration file.
WARNING 0: Region 0 does not need "Radius" defined. Ignoring.
WARNING 1: Actor 1 does not need "Modulation Bits" defined. Ignoring.
Configuration file has 2 warning(s). Press 'Enter' to continue the simulation or 'q'+Enter to quit.

Initializing region parameters.
Number of regions: 1
Initializing microscopic and mesoscopic subvolume parameters.
Number of subvolumes: 1
Finding Neighbours of Each Subvolume...
Number of mesoscopic subvolumes: 0
Initializing mesoscopic subvolumes and meso reaction heap...
Initializing simulation actors...
Number of actors: 2
Number of active actors: 1
Number of passive actors: 1
Simulation output will be written to "results/accord_sample_SEED1.txt".
Simulation summary will be written to "results/accord_sample_SEED1_summary.txt".
Starting up random number generator with seed offset: 1
Starting simulation at 2016-11-03 10:25:18.
Simulation 100.0% complete (1 of 1 repeats). Est. time left: 0 sec.
Ending simulation at 2016-11-03 10:25:18.
Simulation ran in 0.032000 seconds
Writing simulation summary file ...
Memory cleanup ...
Done!

```

Figure 3.3: Sample command line output on Windows running a modified sample configuration file where two unnecessary properties have been added.

2. Every execution uses a seed. A simulation can be called multiple times, each with a different seed, and then all of the realizations in every execution should be independent.

Of course, every execution creates its own output. Why would you want

to use different seeds and create more output files? The main reason is that AcCoRD is a single-threaded program. Multi-threaded execution can be achieved by running multiple instances of the same simulation at the same time, each with its own seed. This concept can be extended to running AcCoRD on a compute cluster (see below). In terms of the number of output files, this is generally not an issue, since the AcCoRD import utility for MATLAB can combine output from a simulation that was run with different seeds.

### 3.4.3 Using a Compute Cluster

If you have access to a computing cluster, then it can be a great resource for running simulations. Clusters might be managed locally, such as in a single research lab or department, or they could be multi-institutional infrastructure, such as the WestGrid and the Centre for Advanced Computing, which have both been used at different stages of AcCoRD's development. Unfortunately, different clusters can have very different ways to set up and execute code, so detailed instructions cannot be provided here and the potential for support by the AcCoRD developer is limited. However, here are some general guidelines and tips to give you an idea of what might be involved:

- You often need some form of SSH access to log in to a remote terminal that serves as the gateway to the cluster. The cluster would usually have some kind of registration procedure that once completed will give you SSH credentials (username/password). On Windows, PuTTY is commonly used for SSH. Linux usually has SSH via command line by default.
- You will need a way to transfer files to and from the cluster. A full install of PuTTY includes utilities to transfer files. Alternatively, an interface-based program like FileZilla can be very helpful to do this.
- You will (probably) need to compile the AcCoRD executable directly on the cluster. Refer to Chapter 2 for instructions on how to compile from source, but replace “preferred directory” with “directory on the cluster” and replace “command line terminal” with “SSH session”. If the cluster is using a Linux-based OS (which it most likely is), then the Linux-based build scripts should work fine.
- A cluster usually has a job submission system where you submit a “job” to run one simulation. The syntax for this can vary greatly. Once you figure out how to submit a job (usually there are sample commands

provided), it can be very helpful to write a script with a for-loop that submits one job for each value in a range of seed values.

- A cluster usually has some kind of fairness protocol to limit how many computing resources a user has access to. You may need to trade off between the number of seeds vs the number of “Repeats” in each simulation, in order to run your simulations in a reasonable time.
- The command line output on a cluster is usually written to a unique data file for each simulation. Generally, there is no mechanism to deal with configuration warnings, so be sure that your configuration file doesn’t generate any warnings (or - **not recommended** - set “Warning Override” to true).
- If you run many simulations with a lot of seed values, then you may need to transfer large numbers of files. It can be much faster to combine files into a “zip” file (or some other compression method) and then transfer the zip file. Total file size can be reduced by up to an order of magnitude, and the transfer rate is much faster when there are fewer individual files.

## 3.5 Understanding AcCoRD Output

This page has instructions for reading and working with AcCoRD output files, including how to import them into MATLAB. These instructions assume that you have already generated the output files by running simulations. If not, then please refer to Section 3.4 for details. Once your output has been imported to MATLAB, you can refer to the AcCoRD Post-Processing page for creating plots and videos. Each AcCoRD simulation generates two output files. The files are named `MY_OUTPUT_SEEDX.txt` and `MY_OUTPUT_SEEDX_summary.txt`, where `MY_OUTPUT` is the “Output Filename” defined in the configuration file, and `X` is the seed value. We will refer to these two files as the **main output file** and the **summary file**, respectively. The summary file is written in JSON format (<http://www.json.org/>). It provides some general information about the simulation, including the run time and the name of the configuration file that was used. Importantly, it also includes information on how to read the main output file. However, **the recommended AcCoRD workflow does not include working directly with these files**. MATLAB utilities are provided which scan the output into structures that are much more convenient to manipulate. If you do not have access to MATLAB, or you need to do some debugging, then information is also provided (in Section 3.5.2) about reading the raw output files.

### 3.5.1 Importing to MATLAB

The recommended method to use AcCoRD output is by importing into MATLAB. AcCoRD comes with utility functions to read output files and facilitate post-processing (see Section 3.6). These functions are all included in the “matlab” directory of the AcCoRD installation. To access these functions, this directory should be your current directory in MATLAB (or, alternatively, on your MATLAB path). **Note:** If the AcCoRD “matlab” directory is not your current directory, then you will also need to include the directory “JSONlab”, which is inside the AcCoRD “matlab” directory, on your MATLAB path. The “JSONlab” directory is only needed when importing (and not post-processing). The primary import function, which you call directly from the MATLAB command line, is called `accordImport.m`. It takes three arguments:

1. **filename** - The prefix of your AcCoRD output files (i.e., without the “\_SEEDX.txt” or “\_SEEDX\_summary.txt” suffixes). This should include a path to the file.

2. **seedRange** - An array listing the seed values associated with the files that you want to import. All files need the same prefix and **should have been created using the same configuration file** (otherwise unexpected behavior may occur).
3. **bWrite** - A switch to write the imported data to a MATLAB “mat” file. If true, then the file filename\_out.mat will be created in the “matlab” directory, where “filename” is the filename argument but without any path.

The accordImport function also has up to two output arguments:

1. **data** - Structure containing the simulation summary and output content.
2. **config** - Structure containing the configuration parameters defined in the configuration file, but only if the configuration file can be found. MATLAB will search for the file as specified in the “ConfigFile” string of the summary file (but **not** including the path). It will search relative to the current directory, and then a number of variations including the parent directory, grandparent directory, and a “config” directory that is either in the current directory or in the parent or grandparent directory. If the file cannot be found, then the “config” output is an empty array.

### The “data” Structure

The fields of the “data” structure are as follows (including indexing information where applicable):

- **“numSeed”** - The number of seed values that were aggregated.
- **“seed”** - The array of seed values that were aggregated.
- **“configName”** - The name of the original configuration file (not including the path)
- **“startTime”** - The computer time at the start of execution (but after loading the configuration file and initializing the system parameters) *of the first seed value.*
- **“endTime”** - The computer time at the end of the simulation (but before memory cleanup) *of the first seed value.*
- **“numRepeatSingle”** - The number of realizations simulated by the first seed.

- **“numRepeat”** - The number total number of realizations by all seeds, *assuming* that the same number of realizations was simulated by each seed.
- **“numActive”** - The number of active actors whose behavior is recorded in the main output files.
- **“activeID”** - Array containing the numeric IDs of the active actors. Numbering of IDs starts at 0 and follows the order of all actors (active and passive) in the configuration file.
- **“activeMaxBits”** - Array containing, for each active actor, the length of the longest bit sequence among all realizations simulated by the first seed.
- **“activeBits”** - Cell array of matrices containing the transmitted bits of each active actor in every realization. Indexing has format activeBits(i,j,k) for the kth observation by the ith active actor in the jth realization.
- **“numPassiveRecord”** - The number of passive actors whose behavior is recorded in the main output files.
- **“passiveRecordID”** - Array containing the numeric IDs of the passive actors. Numbering of IDs starts at 0 and follows the order of all actors (active and passive) in the configuration file.
- **“passiveRecordBTime”** - Array of switches indicating whether the times of the passive actor’s observations were recorded.
- **“passiveRecordMaxCountLength”** - Array containing, for each passive actor, the largest number of observations made by actor in any single realization simulated by the first seed.
- **“passiveRecordNumMolType”** - Array of the number of types of molecules observed by each passive actor.
- **“passiveRecordMolID”** - Cell array of arrays, each of length “passiveRecordNumMolType”, indicating the numeric IDs of the types of molecules that were observed by each passive actor. Numbering of IDs starts at zero. Indexing has format passiveRecordMolID(i,j) for the jth type of molecule observed by the ith passive actor.

- **“passiveRecordBPos”** - Cell array of arrays of switches, each of length “passiveRecordNumMolType”, indicating whether the passive actor recorded the positions of the molecules that were observed. Indexing has format `passiveRecordBPosi(j)` for the *j*th type of molecule observed by the *i*th passive actor.
- **“passiveRecordTime”** - Cell array of matrices, each of size “numRepeat” by “passiveRecordMaxCountLength”, storing the observation times by each passive actor in every realization. Only written to if “passiveRecordBTime” is true. Indexing has format `passiveRecordTimei(j,k)` for the *k*th type of molecule observed by the *i*th passive actor in the *j*th realization.
- **“passiveRecordCount”** - Cell array of 3D arrays, each of size “numRepeat” by “passiveRecordNumMolType” by “passiveRecordMaxCountLength”, storing the number of molecules of each type observed by each passive actor in every realization. Indexing has format `passiveRecordCounti(j,k,m)` for the *m*th observation of the *k*th type of molecule observed by the *i*th passive actor in the *j*th realization.
- **“passiveRecordPos”** - Cell array of cell arrays, each of length “passiveRecordNumMolType”, of 2D cell arrays, each of size “numRepeat” by “passiveRecordMaxCountLength”, of *N* by 3 matrices indicating the coordinates of every molecule observed by each passive actor in every realization. Only written to for elements of “passiveRecordBPos” that are true. Indexing has format `passiveRecordPosijk,m(n,p)` for the *p*th coordinate of the *n*th molecule of the *j*th type observed by the *i*th passive actor in the *m*th observation of the *k*th realization.

### The “config” Structure

The fields of the “config” structure are mostly analogous to those in the corresponding configuration file. The fields match as follows (refer to Section 3.3 for more details about the configuration parameters):

- **“outputFilename”** - “Output Filename”
- **“numRepeatPerSeed”** - “Number of Repeats”
- **“dt”** - “Global Microscopic Time Step”
- **“tFinal”** - “Final Simulation Time”
- **“numMolTypes”** - “Number of Molecule Types”

- “**diffusionCoeff**” - matrix where every row contains the diffusion coefficients for that region. Unless the region defined “**Local Diffusion Coefficients**”, the global “**Diffusion Coefficients**” are used.
- “**numChemRxn**” - length of the “**Chemical Reaction Specification**” array
- “**chemRxn**” - Array of structures corresponding to “**Chemical Reaction Specification**”. The fields of each structure may include (as appropriate):
  - “**rate**” - “**Reaction Rate**”
  - “**label**” - “**Label**”
  - “**bReversible**” - “**Is Reaction Reversible?**”
  - “**revLabel**” - “**Reverse Reaction Label**”
  - “**bSurface**” - “**Surface Reaction?**”
  - “**surfRxnType**” - “**Surface Reaction Type**”
  - “**surfTransProb**” - “**Surface Transition Probability**”
  - “**diffusion**” - “**Surface Reaction Diffusion Coefficient**”
  - “**bEverywhere**” - “**Default Everywhere?**”
  - “**numExceptions**” - length of the “**Exception Regions**” array
  - “**exceptionRegions**” - “**Exception Regions**”
  - “**reactants**” - “**Reactants**”
  - “**products**” - “**Products**”
  - “**bProdReleased**” - “**Products Released?**”
  - “**releaseType**” - “**Release Placement Type**”
- “**subBaseSize**” - “**Subvolume Base Size**”
- “**numRegion**” - length of the “**Region Specification**” array
- “**region**” - Array of structures corresponding to “**Region Specification**”. The fields of each structure may include (as appropriate):
  - “**label**” - “**Label**”
  - “**parent**” - “**Parent label**”
  - “**shape**” - “**Shape**”



- “type” - “Type”
- “surfaceType” - “Surface Type”
- “bMicro” - “Is Region Microscopic?”
- “numSubDim” - “Number of Subvolumes Per Dimension”
- “subvolSizeInt” - “Integer Subvolume Size”
- “radius” - “Radius”
- “anchorCoor” - “Anchor Coordinate”
- “numActor” - length of the “Actor Specification” array
- “numActive” - number of actors in the “Actor Specification” array that are active
- “numPassive” - number of actors in the “Actor Specification” array that are passive
- “actor” - Array of structures corresponding to “Actor Specification”, where only the parameters that are common to both active and passive actors are stored. The fields of each structure may include (as appropriate):
  - “bDefinedByRegions” - “Is Actor Location Defined by Regions?”
  - “numRegion” - length of the “List of Regions Defining Location” array
  - “regionList” - “List of Regions Defining Location”
  - “shape” - “Shape”
  - “boundary” - “Outer Boundary”
  - “bActive” - “Is Actor Active?”
  - “startTime” - “Start Time”
  - “bMaxActions” - “Is There Max Number of Actions?”
  - “maxActions” - “Max Number of Actions”
  - “bIndependent” - “Is Actor Independent?”
  - “actionInterval” - “Action Interval”
  - “bRecord” - “Is Actor Activity Recorded?”

- “**activeID**” - if actor is active, this number indicates the index of the corresponding active actor parameters in the “activeActor” array of structures
- “**passiveID**” - if actor is active, this number indicates the index of the corresponding active actor parameters in the “passiveActor” array of structures
- “**activeActor**” - Array of structures corresponding to the active actors in “**Actor Specification**”. The fields of each structure may include (as appropriate):
  - “**actorID**” - index of the corresponding common actor parameters in the “actor” array of structures.
  - “**bRandNumMolecules**” - “Random Number of Molecules?”
  - “**bRandReleaseTimes**” - “Random Molecule Release Times?”
  - “**releaseInterval**” - “Release Interval”
  - “**slotInterval**” - “Slot Interval”
  - “**bBitsRandom**” - “Bits Random?”
  - “**probBitOne**” - “Probability of Bit 1”
  - “**bitSequence**” - “Bit Sequence”
  - “**numBits**” - length of the “Bit Sequence” array
  - “**modScheme**” - “Modulation Scheme”
  - “**numModBits**” - “Modulation Bits”
  - “**modStrength**” - “Modulation Strength”
  - “**bReleaseType**” - “Is Molecule Type Released?”
- “**passiveActor**” - Array of structures corresponding to the passive actors in “**Actor Specification**”. The fields of each structure include:
  - “**actorID**” - index of the corresponding common actor parameters in the “actor” array of structures.
  - “**bRecordTime**” - “Is Time Recorded with Activity?”
  - “**bRecordMolCount**” - “Is Molecule Type Observed?”
  - “**bRecordMolPosition**” - “Is Molecule Position Observed?”

**Note:** It is generally expected that necessary configuration parameters are defined by the configuration file (i.e., any case that results in a warning about a missing parameter at simulation run time would result in an error here).

### 3.5.2 Reading AcCoRD Output Files

AcCoRD output files are technically readable but are not intended for primary post-processing. The summary file contains two JSON objects and is written so that the **original configuration file is not needed to import the actor behavior to MATLAB** (however, the original configuration file **is needed** to import all of the simulation parameters). The first object has the following fields:

- **“ConfigFile”** - The name of the configuration file used to define the simulation environment (including the path if it was provided).
- **“SEED”** - The seed value used to initialize the random number generator. Does not necessarily match the one in the configuration file since it can be defined at run time.
- **“NumRepeat”** - The number of independent realizations.
- **“StartTime”** - The computer time at the start of execution (but after loading the configuration file and initializing the system parameters).

The second object in the AcCoRD summary file has the following fields:

- **“NumberActiveRecord”** - The number of active actors whose behavior is recorded in the main output file.
- **“ActiveInfo”** - An array of objects, one for each active actor whose behavior was recorded. Each of these objects has the following fields:
  - **“ID”** - The numeric ID of the actor. Numbering starts at 0 and follows the order of all actors (active and passive) in the configuration file.
  - **“MaxBitLength”** - The length of the longest actor bit sequence, among all realizations.
- **“NumberPassiveRecord”** - The number of passive actors whose behavior is recorded in the main output file.
- **“RecordInfo”** - An array of objects, one for each passive actor whose behavior was recorded. Each of these objects has the following fields:

- **“ID”** - The numeric ID of the actor. Numbering starts at 0 and follows the order of all actors (active and passive) in the configuration file.
- **“bRecordTime”** - Switch indicating whether the times of this actor’s observations were recorded.
- **“MaxCountLength”** - The largest number of observations made by the passive actor in any single realization.
- **“NumMolTypeObs”** - The number of types of molecules that the actor observed.
- **“MolObsID”** - The numeric IDs of the types of molecules that were observed. Numbering starts at zero.
- **“bRecordPos”** - Array of switches of length “NumMolTypeObs”. Indicates whether the actor recorded the positions of the molecules that were observed.
- **“EndTime”** - The computer time at the end of the simulation (but before memory cleanup).
- **“RunTime”** - The total run time of the main simulation loop (excludes initialization and cleanup, so should scale linearly with the number of realizations).

The main output file is not written in JSON. Instead, a simple ASCII format is used for fast importing into MATLAB. The structure is as follows:

```

Realization #: (repeats for every realization)
    ActiveActor #: (repeats for every active actor
                    with recorded data)
                    (numbering follows the actor IDs)
                    [Actor bit sequence]
    PassiveActor #: (repeats for every passive actor
                     with recorded data)
                     (numbering follows the actor IDs)
    MolID #: (repeats for every type of
              molecule observed by actor)
    Count:
        [Number of molecules
         in each observation]
    Position: (Only if bRecordPos is true
              for this molecule type)
              [Array of arrays of 3D molecule

```

locations for every observed molecule. Each molecule location is defined within (), and all molecules for a single observation are also within ()]

## 3.6 AcCoRD Post-Processing

This page describes how to use the AcCoRD post-processing utilities that were developed in MATLAB (**for version R2015a or newer**), including functions to plot signal curves and generate videos. These instructions assume that you have already imported simulation output to MATLAB. If not, then please refer to Section 3.5 for details. If you have a configuration file that you want to draw *without simulating it first*, then please refer to Section 3.2 to learn more about using the utility `accordEmptyEnvironment.m`. AcCoRD has two primary post-processing utility files, neither of which you should call directly:

1. **`accordPlotMaker.m`** (the **Plot Maker**) - Plot curves on a figure showing observed signals from one or more realizations of a simulation. Can also plot non-simulation data (in fact, this can be used as a generic plotting function if desired).
2. **`accordVideoMaker.m`** (the **Video Maker**) - Build video files or a sequence of figures showing molecules from a single realization of a simulation. Optional commands enable you to add annotations, a timer, and molecule counters.

Both of these files, which are found in the AcCoRD “matlab” directory, have a wrapper file that you should copy and modify for your specific simulation and plotting requirements. The wrapper files prepare all of the necessary input arguments and include some documentation about how the arguments can be modified. Generally, you should have a different wrapper for every plot or video that you want to make. This makes it easier to re-generate any plot or video at a later time, as needed. The rest of this page will describe how to prepare the wrappers for these two AcCoRD post-processing functions. If you are interested in other post-processing tasks, e.g., calculating bit error rates, then you should refer to Section 3.5 for details on how to read the simulation output after it has been imported.

### 3.6.1 Plot Maker

The default wrapper for plotting curves is the file **`accordPlotMakerWrapper.m`**. To prepare a plot, you should start by copying and renaming this file. By default, it plots the time-varying signal of the number of molecules observed by the passive actor in the default configuration (“`accord.config_sample.txt`”). This is actually rather boring, since the environment is a box with 2 molecules in it and the actor watches the whole box!

The default wrapper is designed to plot one signal from one passive actor in one output file; this can be expanded by adding for loops. The wrapper takes no input arguments, but it has up to two output arguments:

- **“hFig”** - handle to the figure. This can be used to modify the figure after it has been created.
- **“hAxes”** - handle to the figure axes. This can be used to modify the axes (including adding new curves) after it has been created.

The parameters that are defined by the default Plot Maker wrapper are as follows:

- **“fileToLoad”** - Name of the MATLAB “mat” file whose data will be plotted.
- **“hAxes”** - Handle to the figure axes to add the curve. If its value is 0, then a new figure and axes will be created.
- **“customFigProp”** - Structure of figure properties to modify from their AcCoRD defaults. It can be an empty array if there are no properties to change. The file “accordBuildFigureStruct.m” lists some of the structure fields and their default values. Generally, most valid MATLAB figure properties can be modified here (i.e., figure properties that you can change via the “set” command). Examples include the background color and the size of the figure.
- **“customAxesProp”** - Structure of axes properties to modify from their AcCoRD defaults. It can be an empty array if there are no properties to change. The file “accordBuildAxesStruct.m” lists some of the structure fields and their default values. Generally, most valid MATLAB axes properties can be modified here (i.e., axes properties that you can change via the “set” command). Examples include the axis limits and visibility of the grid. The default values are set to favor video generation, so it is recommended to change the “Visible” field to “on” (to show the coordinate axes), the “Projection” field to “orthographic” (which removes depth perception), and the “Clipping” field to “on”.
- **“passiveID”** - Index of passive actor whose signal is to be plotted. The indexing here is from the list of passive actors whose observations were recorded (and **not the underlying actor IDs** in the original simulation).

- **“molID”** - Index of the molecule type whose signal is to be plotted. The indexing here is from the list of molecule types that the passive actor recorded (and **not the underlying molecule type IDs** in the original simulation).
- **“customObsProp”** - Structure of display properties to configure the curve. It can be an empty array if there are no properties to change (default is the time-varying number of molecules observed, averaged over all realizations). There are many different types of plots and settings to configure the curve. For complete details, please refer to the “accord-BuildObserverStruct.m” file. The most critical field of this structure is “obsType”, which defines the type of curve. The options for this field are as follows:
  - **“Sample” (default)** - 2D plot of the time-varying simulation observations. Can be from a specific realization or averaged over any set of realizations (default is to average over all).
  - **“Expected” (non-simulation)** - 2D plot of user-supplied data. Useful for plotting expected curves, but can be any data. Set the “data1” and “data2” fields as vectors of (x, y) data.
  - **“3D Expected” (non-simulation)** - 3D version of “Expected”. Set the “data1”, “data2”, and “data3” fields as matrices of (x, y, z) data.
  - **“Empirical CDF”** - 2D plot of the cumulative distribution function of the simulation observations, i.e., measures the probability that an observation will be less than or equal to some value. Includes all observations at all observation times into one CDF.
  - **“3D Empirical CDF”** - time-varying version of “Empirical CDF”. Every observation time has its own corresponding CDF.
  - **“Expected CDF” (non-simulation)** - 2D plot of the cumulative distribution function for the specified probability distribution. Options are based on the Binomial distribution, with a corresponding number of trials “numTrials” and success probability “trialProbability”. The other distributions are the Poisson and Gaussian approximations of the Binomial distribution.
  - **“3D Expected CDF” (non-simulation)** - 3D version of “Expected CDF”, where the “trialProbability” is a vector instead of a scalar and the “data2” field lists event times associated with the each trial probability.



- **“Empirical PMF”** - 2D plot of the probability mass function of the simulation observations, i.e., measures the probability that an observation will be equal to some value. Includes all observations at all observation times into one PMF.
- **“3D Empirical PMF”** - time-varying version of “Empirical PMF”. Every observation time has its own corresponding PMF.
- **“Expected PMF” (non-simulation)** - 2D plot of the probability mass function for the specified probability distribution. Options are based on the Binomial distribution, with a corresponding number of trials “numTrials” and success probability “trialProbability”. The other distributions are the Poisson and Gaussian approximations of the Binomial distribution.
- **“3D Expected PMF” (non-simulation)** - 3D version of “Expected PMF”, where the “trialProbability” is a vector instead of a scalar and the “data2” field lists event times associated with the each trial probability.
- **“Mutual Information”** - 2D plot of mutual information between simulation observations relative to a reference observation, which tells you how much the knowledge of the reference observation helps in predicting the value of other observations. The reference observation is the first one specified. Generally, mutual information should decrease as observations become more separated in time. It is measured using all realizations.
- **“3D Mutual Information”** - 3D version of “Mutual Information”, where there are multiple reference observations. The “data1” vector defines what (relative) observations to measure from each reference. If a value in “data1” is 0, then the corresponding surface points will measure the entropy of each reference observation.
- **“Monte Carlo Mutual Information” (non-simulation)** - 2D plot of mutual information between independent random variables generated from Binomial distributions with different success probabilities. Since there is a finite number of realizations of the random variables, the mutual information will have a non-zero value (even though the true mutual information between independent random variables is 0). This kind of curve can take a long time to generate.
- **“3D Monte Carlo Mutual Information” (non-simulation)** - 3D version of “Monte Carlo Mutual Information”, where there

is a matrix instead of a vector of success probabilities. This kind of curve can take a very long time to generate.

- **“customCurveProp”** - Structure of curve (2D) or surface (3D) properties to modify from their AcCoRD defaults. It can be an empty array if there are no properties to change. The file “accordBuildCurveStruct.m” lists some of the structure fields and their default values for 2D plots. The file “accordBuildSurfStruct.m” lists some of the structure fields and their default values for 3D plots. Generally, most valid MATLAB curve and surface properties can be modified here (i.e., curve or surface properties that you can change via the “set” command). Examples include marker size, line style, and display name (which you see if you turn on the legend).

Once the wrapper is configured as desired, you can run it to run the Plot Maker and generate the figure. Complete examples of modified wrapper files for generating plots can be found on the AcCoRD Examples webpage (<https://warwick.ac.uk/fac/sci/eng/staff/ajgn/software/accord/examples/>).

### 3.6.2 Video Maker

Before running simulations to create videos, you should consider the following guidelines:

- Generally, the simulations that you run to create videos may not be the same as those for other post-processing. This is because you probably want passive actors to observe more of the environment, and they need exact molecule locations. Recording molecule locations results in larger output files and takes longer to import into MATLAB.
- In order to see molecules in a video, their locations need to be recorded by a passive actor. If not, then there will be nothing to see! So, if you want to watch molecules everywhere in the environment (and in an isolated location, e.g., at an intended receiver), then you will need to configure one or more passive actors to occupy the entire simulation environment. One reason for multiple passive actors that do not overlap is so that you can use different colors to indicate when molecules have entered different areas of the environment.
- A video will only display a single realization, so a simulation run to create a video will mostly likely only need one realization (unless you

want to try multiple realizations and pick one with the observations that you want to use).

- The display of any actor or region in the environment is optional and independent of what molecules are being drawn. You can have molecules moving in an environment that appears to be empty.

### The Video Wrapper File

The default wrapper for creating videos is the file **accordVideoMaker-Wrapper.m**. To prepare a video, you should start by copying and renaming this file. By default, it makes a video of the two molecules diffusing in the default configuration (“accord\_config\_sample.txt”). The wrapper takes no input arguments, but it has up to two output arguments:

- **“hFig”** - Array of handles to the figure. This can be used to modify the figures after they have been created (there is an option to create multiple figures instead of building a video).
- **“hAxes”** - Array of handles to the figure axes. This can be used to modify the axes after they have been created.

The parameters that are defined by the default Video Maker wrapper are as follows (you can see that a number of them are the same as those for the Plot Maker wrapper):

- **“fileToLoad”** - Name of the MATLAB “mat” file whose data will be plotted.
- **“bMakeVideo”** - Switch to control whether a video is generated. If false, then each observation will be plotted as a separate figure. If true (or a positive integer), then all observations are plotted in the same figure and stitched into a video file. Entering a number greater than 1 will repeat each frame the specified number of times (to decrease the apparent frame rate).
- **“videoName”** - Filename to save the video file to. The relevant extension will be added automatically.
- **“videoFormat”** - Format of the video. See the MATLAB documentation for the “VideoWriter” function to see the full list of options. “Motion JPEG AVI” creates a high quality video with a large file size (but which can be easily converted to a high quality mp4 with a much smaller file size with free tools like VLC Media Player

(<https://www.videolan.org/>). To generate a small file directly in Windows, set to “MPEG-4” (though experience has shown much better results by building an avi file first and then converting with VLC).

- **“curRepeat”** - Index of the simulation realization to use.
- **“scale”** - A scaling factor used to mitigate display problems observed when trying to draw objects that have very small dimensions. This factor is applied to region and actor coordinates. It is recommended that the smallest object (that isn’t a molecule) have a length on the order of 1, so a system defined on the order of microns should set “scale” to 1e6.
- **“observationToPlot”** - Array of indices listing the observations that will be recorded and in what order. Actual actor start times and action intervals are ignored; it is assumed that all recorded actors make observations at the same times. Indices that are larger than the maximum number of observations in the simulation are just ignored.
- **“customFigProp”** - Structure of figure properties to modify from their AcCoRD defaults. It can be an empty array if there are no properties to change. The file “accordBuildFigureStruct.m” lists some of the structure fields and their default values. Generally, most valid MATLAB figure properties can be modified here (i.e., figure properties that you can change via the “set” command). Examples include the background color and the size of the figure.
- **“customAxesProp”** - Structure of axes properties to modify from their AcCoRD defaults. It can be an empty array if there are no properties to change. The file “accordBuildAxesStruct.m” lists some of the structure fields and their default values. Generally, most valid MATLAB axes properties can be modified here (i.e., axes properties that you can change via the “set” command). Examples include the axis limits and visibility of the grid. The default values are set to favor video generation, so you may not need to change any values from their defaults.
- **“customVideoProp”** - Structure of video properties to modify from their AcCoRD defaults. It can be an empty array if there are no properties to change. The file “accordInitializeVideo.m” lists some of the structure fields and their default values. Generally, most valid MATLAB video object properties can be modified here (i.e., video properties that you can change via the “set” command). Examples include the

frame rate and the video quality (although video quality does not apply to lossless formats).

- **“regionToPlot”** - Array of indices of the regions that will be plotted. This can be an empty array if no regions are to be drawn. Choosing to draw a region is independent of choosing to draw the molecules that are inside that region.
- **“customRegionProp”** - Structure of region display properties to modify from their AcCoRD defaults. It can be an empty array if there are no properties to change. The file “accordBuildDispStruct.m” lists the structure fields and their default values, which do **not** correspond to a specific set of MATLAB properties. Properties include color and opacity.
- **“actorToPlot”** - Array of indices of the actors that will be plotted. Indexing is from the initial actor list (of both active and passive actors). This can be an empty array if no actors are to be drawn. Choosing to draw a actor is independent of choosing to draw the molecules that are inside that actor. Actors are drawn after regions, so if an actor is defined by regions then it will be drawn on top of its regions.
- **“customActorProp”** - Structure of actor display properties to modify from their AcCoRD defaults. It can be an empty array if there are no properties to change. The file “accordBuildDispStruct.m” lists the structure fields and their default values, which do **not** correspond to a specific set of MATLAB properties. Properties include color and opacity.
- **“passiveActorToPlot”** - Array of indices of passive actors whose molecules are to be plotted. Indexing is from the list of passive actors whose observations were recorded. Actors listed here will have their observed molecules plotted and not the actor shapes themselves.
- **“molToPlot”** - Cell array of arrays of indices of molecule types to plot. Each cell corresponds to the passive actor index specified in “passiveActorToPlot”. The array in a given cell lists the indices of the molecule types of that actor that are to be drawn, where the indexing corresponds only to the molecule types that the specific actor observed. For example, if a simulation used 4 types of molecules, and an actor only observed the 4th type of molecule, then its corresponding array can only be [1].

- **“customMolProp”** - Cell array of structures of molecule display properties to modify from their AcCoRD defaults. It can be an empty array if there are no properties to change. Indexing in this cell array matches the “molToPlot” cell array. The file “accordBuildMarkerStruct.m” lists the structure fields and their default values, which do **not** correspond to a specific set of MATLAB properties.
- **“cameraAnchorArray”** - Cell array of cell arrays defining anchor points for the camera display. It can be an empty cell array if the camera view will not be modified. You can control how the camera behaves during a video such that it moves between anchor locations that are defined for specific frames. Each anchor point is a cell array that defines a complete set of camera settings, in the format ‘CameraPosition’, ‘CameraTarget’, ‘CameraViewAngle’, ‘CameraUpVector’. See the MATLAB camera documentation for more details. You can also add camera anchors to this cell array via the command line at run time (see later in this Section).
- **“frameCameraAnchor”** - Array that specifies the camera anchor (from the “cameraAnchorArray” cell array or defined at run time) used in each frame. Length of this array should be equal to that of “observationToPlot”. It can be an empty array if “cameraAnchorArray” is also empty. Values in the array must be 0 or match the index of of anchor points in “cameraAnchorArray”. If a frame has value 0, and there are anchor points defined for an earlier and a later frame, then the camera settings for the current frame will be linearly interpolated between the camera settings for the surrounding anchors. You can also modify this array via the command line at run time (see later in this Section).

Once the wrapper is configured as desired, you can run it. See the following section for more details.

### Running the Video Maker

The wrapper function is run without any import arguments (unless you added them to your copy). When it calls the Video Maker, a figure is generated and the environment is plotted according to the wrapper arguments (without any molecules yet). If a video file is to be generated, then it is initialized (and over-writing the file if it already existed). Then, execution enters MATLAB’s “debug mode” so that you can do any of the following:

1. **Change how the regions and actors appear.** If you are making a

video (and **not** a sequence of images), then changing the appearance of any regions or actors (e.g., color, line width) will apply to the video.

## 2. Modify the camera settings.

- Call the function “`accordAddCameraAnchor(FRAME_INDICES)`” to add a new camera anchor to the “`cameraAnchorArray`” cell array, where the current camera settings are used to define a camera anchor point for the frames defined by the `FRAME_INDICES` array.
- If the “`cameraAnchorArray`” cell array defined in the wrapper file was originally empty, then you will see a default view. You can modify the camera as desired (i.e., by panning, zooming, rotating, etc.) and it will be applied as-is for all of the video or sequence of images.
- If the “`cameraAnchorArray`” cell array was not originally empty, then the camera settings will show the first anchor point. Changes made to the camera will only apply if you add new anchors.
- Calling “`accordAddCameraAnchor`” without any arguments will delete all existing anchors and creates an anchor for every frame using the current camera settings.

## 3. Add text or graphic annotations.

You can add custom text or other annotations that will appear on-screen for any specified subset of the frames. Once you have created the desired annotations for a particular subset of frames, call “`accordAddAnnotation(FRAME_INDICES)`” to save the annotations, which will appear for the frames defined by the `FRAME_INDICES` array. Here are some tips and guidelines:

- De-select the “Edit Plot” button in the figure toolbar **before** calling `accordAddAnnotation`. If not, then the last selected object will have the selection highlighting included in the annotation capture. This highlighting will be visible in the final video or image sequence.
- Generally, annotations will be drawn on top of the molecules.
- Any annotations that are still on screen when you resume execution of the Video Maker will remain in all frames.
- Simulation timers and molecule counters (see below) are all ignored when you call `accordAddAnnotation`.

4. **Add a simulation timer.** Call the function “`accordAddTimeDisplay(NUM_DECIMAL_PLACES)`” to print dynamic simulation time in seconds, where `NUM_DECIMAL_PLACES` is the (optional) number of decimal places. The default is 4 places. The timer text that appears can be moved or re-configured (e.g., font size) as desired.
5. **Add a molecule counter.** Call the function “`accordAddObservationCount(PASSIVE_ACTOR_INDEX, MOLECULE_INDEX, TEXT)`” to print a molecule counter. `PASSIVE_ACTOR_INDEX` is the index of the passive actor to count, from the list of passive actors whose observations were saved to output. `MOLECULE_INDEX` is the index of molecule to count, from the passive actor’s molecule list. `TEXT` is prefix text that will appear before the counter value. The counter text that appears can be moved or re-configured (e.g., font size) as desired.

To resume execution and generate the video (or sequence of images), enter “`dbcont`” at the command line. If you want to cancel execution, enter “`dbquit`”.



## Chapter 4

# AcCoRD Configuration Examples

This Chapter contains descriptions of configuration and post-processing file examples to run in the AcCoRD simulator. To run these examples yourself, you will need to download and install AcCoRD (see Chapter 2). For more information on how to use these files, you can refer to Chapter 3. The examples are sorted into two main categories. The configuration files of the **latest release examples** are included with every download of AcCoRD and are kept up to date to run on the latest release. The **specific release examples** are maintained for a specific version and are intended to recreate results from a specific publication or demonstrate a new feature associated with that release. The files for the specific version examples can be found on the AcCoRD Examples webpage (<https://warwick.ac.uk/fac/sci/eng/staff/ajgn/software/accord/examples/>).

### 4.1 Latest Release Examples

The following configuration files are kept up to date for the latest version of AcCoRD. To limit execution times, most of these simulations only run a single realization. To access these files, you can download the latest version of AcCoRD (<https://warwick.ac.uk/fac/sci/eng/staff/ajgn/software/accord/downloads/>; see Chapter 2).

- “**accord\_config\_sample.txt**” - Places 2 molecules in a microscopic box and tracks their locations as they diffuse. No reactions. **This is one of only two sample configuration files that track molecule locations.**

- **“accord\_config\_sample\_all\_shapes\_hybrid.txt”** - Places molecules uniformly in a large box that is divided into a mix of microscopic and mesoscopic regions. Demonstrates the nesting of regions. No reactions.
- **“accord\_config\_sample\_all\_shapes\_meso.txt”** - Places molecules uniformly in a large box that is divided into a number of mesoscopic regions. Demonstrates the nesting of regions. No reactions.
- **“accord\_config\_sample\_all\_shapes\_micro.txt”** - Places molecules uniformly in a large box that is divided into a number of microscopic regions. Demonstrates the nesting of regions. No reactions.
- **“accord\_config\_sample\_communication.txt”** - Small spherical source releases finite-width pulses of molecules that are observed as they pass through a transparent spherical receiver. Environment is all microscopic. The transmitter modulates its signal according to a pre-defined sequence of bits. Each symbol has two bits. The symbols (0, 1, 2, 3) result in (0, 1000, 2000, 3000) molecules being released over a release interval of 1ms, which is 10% of the symbol interval. No reactions.
- **“accord\_config\_sample\_communication\_chemical.txt”** - Point source releases finite-width pulses of molecules that can bind to a spherical receiver. Environment is all microscopic. Transmitter modulates its signal according to a randomly-generated bit sequence. Each symbol has 2 bits (values are 0, 1, 2, 3). The value of the symbol is multiplied by a stochastic generation rate of 120000 molecules per second that is active for 1ms of the transmitter’s 10ms symbol interval. Molecules can reversibly bind to the surface of the receiver.
- **“accord\_config\_sample\_crowding.txt”** - Places 10 molecules in a microscopic box that has a reflecting box nested inside. A bimolecular reaction with a finite unbinding radius maintains a minimum distance between pairs of molecules.
- **“accord\_config\_sample\_flow.txt”** - Point source releases an impulse of molecules that are counted inside a spherical observer. There is a net flow in the direction from the source to the observer, and the flow has a slower velocity within the observer. Environment is all microscopic.
- **“accord\_config\_sample\_flow\_closed\_hybrid.txt”** - Places molecules inside a pipe that is closed to form a square. Each side of the pipe has a different flow direction, such that the molecules

undergo a net counterclockwise motion. Two of the sides are mesoscopic and two of the sides are microscopic. **This is one of only two sample configuration files that track molecule locations.**

- **“accord\_config\_sample\_hybrid.txt”** - Places molecules in a box that is one half microscopic and one half mesoscopic. The mesoscopic region has 125 subvolumes.
- **“accord\_config\_sample\_pipe\_reaction\_diffusion.txt”** - Places molecules at one end of a rectangular pipe. An absorbing region with a finite absorption rate is at the other end of the pipe. The pipe is mesoscopic.
- **“accord\_config\_sample\_pipe\_reaction\_diffusion\_microscopic.txt”** - Places molecules at one end of a rectangular pipe. An absorbing region with a finite absorption rate is at the other end of the pipe. The pipe is microscopic.
- **“accord\_config\_sample\_point\_diffusion.txt”** - Point source releases an impulse of molecules that are observed by various transparent receivers, including a sphere and a box. Environment is all microscopic. No reactions.
- **“accord\_config\_sample\_reactor.txt”** - Creates and destroys molecules in a mesoscopic box. On average, one molecule is created every second and has an expected lifetime of 100 seconds.
- **“accord\_config\_sample\_reactor\_2nd\_order.txt”** - Places molecules in a mesoscopic box. The molecules can degrade by participating in an enzyme kinetic reaction. The diffusion coefficients of all molecule are set to zero but there is only one subvolume so no diffusion is needed (molecules are always assumed to have a random location inside a mesoscopic subvolume).
- **“accord\_config\_sample\_reactor\_microscopic.txt”** - Creates and destroys molecules in a microscopic box. On average, one molecule is created every second and has an expected lifetime of 100 seconds.
- **“accord\_config\_sample\_surface.txt”** - Places molecules of many types in a microscopic box. Some types of molecules can pass through one face of the box into an adjacent box, and some types of molecules can be absorbed by one face of the box (either reversible or irreversibly). This environment re-creates that used for Figure 6a in [4], a paper by

Steven S. Andrews where he derived the surface reaction probabilities for Smoldyn (<http://www.smoldyn.org/>). These probabilities are also implemented in AcCoRD.

## 4.2 Specific Release Examples

The following configuration files were written for a specific version of AcCoRD. Old versions of AcCoRD can be found on Github (<https://github.com/adamjgnoel/AcCoRD/releases>).

### 4.2.1 Sample Videos in the AcCoRD Journal Paper

[1] has a series of 8 videos in its supplementary materials. These videos can also be found on Youtube (<https://www.youtube.com/playlist?list=PLZ7uYXG-7XF8UyhFrIuQIiZig1XA89e3i>). The simulations were run using version 0.7 and 0.7.0.1, and the videos were generated using version 1.0, but all can be re-created using just 1.0.

- The corresponding configuration directory also includes a MATLAB m-file (“accord\_config\_journal\_video\_1\_draw\_environment.m”) that will draw the regions defined for the first video without needing to run a simulation. This script should be placed in the AcCoRD “matlab” directory to run properly.
- A separate directory includes the Video Maker wrappers that can be used to build the videos once the simulation output has been imported into MATLAB.

# Bibliography

- [1] A. Noel, K. C. Cheung, R. Schober, D. Makrakis, and A. Hafid, “Simulating with AcCoRD: Actor-based communication via reaction–diffusion,” *Nano Communication Networks*, vol. 11, pp. 44–75, Mar. 2017.
- [2] A. Noel, K. C. Cheung, and R. Schober, “On the statistics of reaction-diffusion simulations for molecular communication,” in *Proc. ACM NANOCOM*, Sep. 2015, pp. 1–6.
- [3] —, “Multi-scale stochastic simulation for diffusive molecular communication,” in *Proc. IEEE ICC*, Jun. 2015, pp. 1109–1115.
- [4] S. S. Andrews, “Accurate particle-based simulation of adsorption, desorption and partial transmission,” *Physical biology*, vol. 6, no. 4, p. 046015, 2009.